305-CD-049-001

EOSDIS Core System Project

Flight Operations Segment (FOS) Data Management Design Specification for the ECS Project

October 1995

Hughes Information Technology Corporation
Upper Marlboro, MD

Flight Operations Segment (FOS)
Data Management Program Design Specification
for the ECS Project

October 1995

Prepared Under Contract NAS5-60000
CDRL Item #046

APPROVED BY

Cal Moore /s/ 9/29/95

Calvin Moore, FOS CCB Chairman Date
EOSDIS Core System Project

Hughes Information Technology Corporation
Upper Marlboro, Maryland

This page intentionally left blank.

Preface

This document, one of nineteen, comprises the detailed design specification of the FOS subsystems for Releases A and B of the ECS project. This includes the FOS design to support the AM-1 launch.

The FOS subsystem design specification documents for Releases A and B of the ECS project include:

305-CD-040	FOS Design Specification (Segment Level Design)
305-CD-041	Planning and Scheduling Design Specification
305-CD-042	Command Management Design Specification
305-CD-043	Resource Management Design Specification
305-CD-044	Telemetry Design Specification
305-CD-045	Command Design Specification
305-CD-046	Real-Time Contact Management Design Specification
305-CD-047	Analysis Design Specification
305-CD-048	User Interface Design Specification
305-CD-049	Data Management Design Specification
305-CD-050	Planning and Scheduling PDL
305-CD-051	Command Management PDL
305-CD-052	Resource Management PDL
305-CD-053	Telemetry PDL
305-CD-054	Real-Time Contact Management PDL
305-CD-055	Analysis PDL
305-CD-056	User Interface PDL
305-CD-057	Data Management PDL
305-CD-058	Command PDL

Object models presented in this document have been exported directly from CASE tools and in some cases contain too much detail to be easily readable within hard copy page constraints. The reader is encouraged to view these drawings on line using the Portable Document Format (PDF) electronic copy available via the ECS Data Handling System (EDHS) at URL http://edhs1.gsfc.nasa.gov.

This document is a contract deliverable with an approval code 2. As such, it does not require formal Government approval, however, the Government reserves the right to request changes within 45 days of the initial submittal. Once approved, contractor changes to this document are handled in accordance with Class I and Class II change control requirements described in the EOS Configuration Management Plan, and changes to this document shall be made by document change notice (DCN) or by complete revision.

Any questions should be addressed to:

Data Management Office
The ECS Project Office
Hughes Information Technology Corporation
1616 McCormick Drive
Upper Marlboro, MD 20774-5372

Abstract

The FOS Design Specification consists of a set of 19 documents that define the FOS detailed design. The first document, the FOS Segment Level Design, provides an overview of the FOS segment design, the architecture, and analyses and trades. The next nine documents provide the detailed design for each of the nine FOS subsystems. The last nine documents provide the PDL for the nine FOS subsystems. It also allocates the level 4 FOS requirements to the subsystem design.

Keywords: FOS, design, specification, analysis, IST, EOC

This page intentionally left blank.

Change Information Page

List of Effective Pages	
Page Number	**Issue**
Title	Original
iii through xiv	Original
1 -1 and 1-2	Original
2-1 through 2-4	Original
3-1 through 3-164	Original
AB-1 through AB-8	Original
GL-1 thtough GL-8	Original

Document History			
Document Number	**Status/Issue**	**Publication Date**	**CCR Number**
305-CD-049-001	Original	October 1995	95-0654

This page intentionally left blank.

Contents

Preface

Abstract

Change Information Page

1. Introduction

2. Related Documentation

3. Data Management Subsystem

Abbreviations and Acronyms

Glossary

Figures

Tables

This page intentionally left blank.

1. Introduction

1.1 Identification

The contents of this document defines the design specification for the Flight Operations Segment (FOS). Thus, this document addresses the Data Item Description (DID) for CDRL Item 046 305/DV2 under Contract NAS5-60000.

1.2 Scope

The Flight Operations Segment (FOS) Design Specification defines the detailed design of the FOS. It allocates the level 4 FOS requirements to the subsystem design. It also defines the FOS architectural design. In particular, this document addresses the Data Item Description (DID) for CDRL # 053, the Segment Design Specification.

This document reflects the August 23, 1995 Technical Baseline maintained by the contractor configuration control board in accordance with ECS Technical Direction No. 11, dated December 6, 1994. It covers releases A and B for FOS. This corresponds to the design to support the AM-1 launch.

1.3 Purpose

The FOS Design Specification consists of a set of 19 documents that define the FOS detailed design. The first document, the FOS Segment Level Design, provides an overview of the FOS segment design, the architecture, and analyses and trades. The next nine documents provide the detailed design for each of the nine FOS subsystems. The last nine documents provide the PDL for the nine FOS subsystems.

1.4 Status and Schedule

This submittal of DID 305/DV2 incorporates the FOS detailed design performed during the Critical Design Review (CDR) time frame. This document is under the ECS Project configuration control.

1.5 Document Organization

305-CD-040 contains the overview, the FOS segment models, the FOS architecture, and FOS analyses and trades performed during the design phase.

305-CD-041 contains the detailed design for Planning and Scheduling Design Specification.

305-CD-042 contains the detailed design for Command Management Design Specification.

305-CD-043 contains the detailed design for Resource Management Design Specification.

305-CD-044 contains the detailed design for Telemetry Design Specification.

305-CD-045 contains the detailed design for Command Design Specification.

305-CD-046 contains the detailed design for Real-Time Contact Management Design Specification.

305-CD-047 contains the detailed design for Analysis Design Specification.

305-CD-048 contains the detailed design for User Interface Design Specification.

305-CD-049 contains the detailed design for Data Management Design Specification.

305-CD-050 contains Planning and Scheduling PDL.

305-CD-051 contains Command Management PDL.

305-CD-052 contains Resource Management PDL.

305-CD-053 contains the Telemetry PDL.

305-CD-054 contains the Real-Time Contact Management PDL.

305-CD-055 contains the Analysis PDL.

305-CD-056 contains the User Interface PDL.

305-CD-057 contains the Data Management PDL.

305-CD-058 contains the Command PDL.

Appendix A of the first document contains the traceability between Level 4 Requirements and the design. The traceability maps the Level 4 requirements to the objects included in the subsystem object models.

Glossary contains the key terms that are included within this design specification.

Abbreviations and acronyms contains an alphabetized list of the definitions for abbreviations and acronyms used within this design specification.

2. Related Documentation

2.1 Parent Document

The parent documents are the documents from which this FOS Design Specification's scope and content are derived.

194-207-SE1-001	System Design Specification for the ECS Project
304-CD-001-002	Flight Operations Segment (FOS) Requirements Specification for the ECS Project, Volume 1: General Requirements
304-CD-004-002	Flight Operations Segment (FOS) Requirements Specification for the ECS Project, Volume 2: AM-1 Mission Specific

2.2 Applicable Documents

The following documents are referenced within this FOS Design Specification or are directly applicable, or contain policies or other directive matters that are binding upon the content of this volume.

194-219-SE1-020	Interface Requirements Document Between EOSDIS Core System (ECS) and NASA Institutional Support Systems
209-CD-002-002	Interface Control Document Between EOSDIS Core System (ECS) and ASTER Ground Data System, Preliminary
209-CD-003-002	Interface Control Document Between EOSDIS Core System (ECS) and the EOS-AM Project for AM-1 Spacecraft Analysis Software, Preliminary
209-CD-004-002	Data Format Control Document for the Earth Observing System (EOS) AM-1 Project Data Base, Preliminary
209-CD-025-001	ICD Between ECS and AM1 Project Spacecraft Software Development and Validation Facilities (SDVF)
311-CD-001-003	Flight Operations Segment (FOS) Database Design and Database Schema for the ECS Project
502-ICD-JPL/GSFC	Goddard Space Flight Center/MO&DSD, Interface Control Document Between the Jet Propulsion Laboratory and the Goddard Space Flight Center for GSFC Missions Using the Deep Space Network
530-ICD-NCCDS/MOC	Goddard Space Flight Center/MO&DSD, Interface Control Document Between the Goddard Space Flight Center Mission Operations Centers and the Network Control Center Data System
530-ICD-NCCDS/POCC	Goddard Space Flight Center/MO&DSD, Interface Control Document Between the Goddard Space Flight Center Payload Operations Control Centers and the Network Control Center Data System

530-DFCD-NCCDS/POCC	Goddard Space Flight Center/MO&DSD, Data Format control Document Between the Goddard Space Flight Center Payload Operations Control Centers and the Network Control Center Data System
540-041	Interface Control Document (ICD) Between the Earth Observing System (EOS) Communications (Ecom) and the EOS Operations Center (EOC), Review
560-EDOS-0230.0001	Goddard Space Flight Center/MO&DSD, Earth Observing System (EOS) Data and Operations System (EDOS) Data Format Requirements Document (DFRD)
ICD-106	Martin Marietta Corporation, Interface Control Document (ICD) Data Format Control Book for EOS-AM Spacecraft
none	Goddard Space Flight Center, Earth Observing System (EOS) AM-1 Flight Dynamics Facility (FDF) / EOS Operations Center (EOC) Interface Control Document

2.3 Information Documents

2.3.1 Information Document Referenced

The following documents are referenced herein and, amplify or clarify the information presented in this document. These documents are not binding on the content of this FOS Design Specification.

194-201-SE1-001	Systems Engineering Plan for the ECS Project
194-202-SE1-001	Standards and Procedures for the ECS Project
193-208-SE1-001	Methodology for Definition of External Interfaces for the ECS Project
308-CD-001-004	Software Development Plan for the ECS Project
194-501-PA1-001	Performance Assurance Implementation Plan for the ECS Project
194-502-PA1-001	Contractor's Practices & Procedures Referenced in the PAIP for the ECS Project
604-CD-001-004	Operations Concept for the ECS Project: Part 1-- ECS Overview, 6/95
604-CD-002-001	Operations Concept for the ECS project: Part 2B -- ECS Release B, Annotated Outline, 3/95
604-CD-003-001	ECS Operations Concept for the ECS Project: Part 2A -- ECS Release A, Final, 7/95
194-WP-912-001	EOC/ICC Trade Study Report for the ECS Project, Working Paper
194-WP-913-003	User Environment Definition for the ECS Project, Working Paper
194-WP-920-001	An Evaluation of OASIS-CC for Use in the FOS, Working Paper
194-TP-285-001	ECS Glossary of Terms
222-TP-003-006	Release Plan Content Description

none	Hughes Information Technology Company, Technical Proposal for the EOSDIS Core System (ECS), Best and Final Offer
560-EDOS-0211.0001	Goddard Space Flight Center, Interface Requirements Document (IRD) Between the Earth Observing System (EOS) Data and Operations System (EDOS), and the EOS Ground System (EGS) Elements, Preliminary
NHB 2410.9A	NASA Hand Book: Security, Logistics and Industry Relations Division, NASA Security Office: Automated Information Security Handbook

This page intentionally left blank.

3. Data Management Subsystem

The Data Management Subsystem (DMS) provides services for database update and retrieval, file and table management, external interfaces, and data archival and retrieval. The DMS provides the capability to update the Project Database with the spacecraft definitions and the instrument definitions. The DMS generates an operational database from the Project Database. The DMS provides services to all FOS subsystems for retrieval of the operational database. The DMS provides file and table management services so that application software will have the capability to store and retrieve data files, and add, update, delete and retrieve from database tables. The DMS provides an interface to FDF, EDOS, and SCDO. The DMS provides services for archiving and retrieving telemetry data, and events data.

3.1 Data Management Subsystem Context Diagram

The DMS interfaces with the other FOS subsystems and with external entities. These interfaces are shown in Figure 3.1-1.

User Interface Subsystem - The FOS User Interface Subsystem interfaces with the DMS when retrieving format definitions, procedures, reports, event history, templates, and other data files. The DMS receives request for data files, event history requests, analysis requests, and replay requests from the User Interface Subsystem. User Interface Subsystem sends procedures, templates, and definitions to the DMS for storage.

Spacecraft and Instrument Manufacturer - The Spacecraft and Instrument Manufacturer provide the spacecraft and instrument definitions to the DMS. Technical documentation about the spacecraft and instruments are stored by the DMS.

SCDO Ingest and Data Server - The DMS sends data to the SCDO Ingest for long term storage, and retrieves long term data from the SCDO Data Server.

Resource Management Subsystem - The Resource Management Subsystem interfaces with the DMS when requesting default configuration procedure, and database ids. The database ids are used when retrieving a database during replay of telemetry. The DMS provides the database ids and default configuration procedures to the Resource Management Subsystem.

Real-Time Contact Manager - The DMS receives Nascom blocks, performance data, and events from the Real-Time Contact Manager. The data is made available by the DMS.

Analysis Subsystem - The DMS provides historical telemetry data, and telemetry databases to the Analysis subsystem. The Analysis Subsystem needs limits, calibration curves, and analysis algorithms from the telemetry database so that statistics can be generated from the telemetry data. The DMS also provides FDF Orbital Information to the Analysis for statistics purposes. Analysis results generated from telemetry data and FDF Orbital Information are stored by the DMS and are made available by the DMS for quick access. Analysis events are stored by the DMS.

SCDO Management Subsystem - The DMS sends status to the SCDO Management Subsystem. The status contains information about the configuration and state of application software in the DMS. The DMS receives events from the SCDO Management Subsystem.

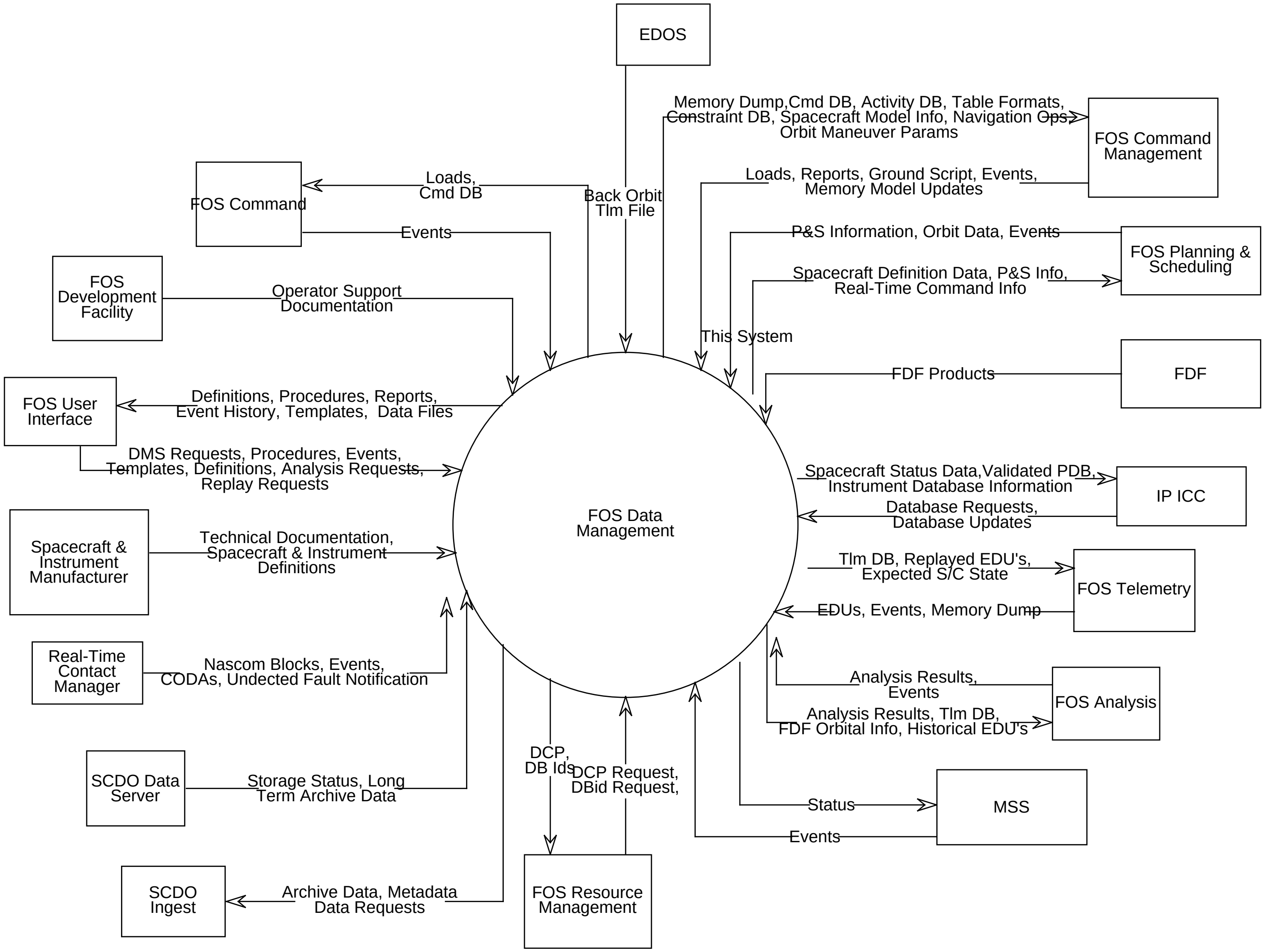


Figure 3.1-1. DMS Context Diagram

Telemetry Subsystem - The DMS provides the telemetry database to the Telemetry Subsystem. The Telemetry Subsystem needs the telemetry database when decommutating real-time and replay telemetry. The Telemetry Subsystem sends real-time housekeeping telemetry, memory dumps, and telemetry events to the DMS for storage.

IP ICC - The DMS receives database requests and database updates from an IP ICC, sends spacecraft status data and database information to an IP ICC.

Planning and Scheduling Subsystem - The DMS provides the spacecraft definitions database and planning and scheduling information to the Planning and Scheduling Subsystem. The Planning and Scheduling Subsystem uses the activity and constraint definitions from the spacecraft definitions database. The DMS provides storage for the orbital data that Planning and Scheduling. Planning and Scheduling events are stored by the DMS.

Command Management Subsystem - The DMS provides the command database, activity database, constraint database and files used to support planned operations to the Command Management subsystem. Activity definitions, constraint definitions and command definitions from the database are used when generating command loads. The DMS provides a storage area for command loads, memory dumps, and ground scripts. Command Management events and reports are stored by the DMS.

Command Subsystem - The DMS provides the command database to the Command subsystem. The database is used during a real-time contact to build commands to be uplinked to the spacecraft. The DMS provides previously generated loads (e.g., microprocessor memory loads) from the FOS file storage area to the Command subsystem for uplink to the spacecraft. Command events generated by the Command subsystem are stored by the DMS.

FOS Development Facility - The Operator Support Documentation generated by the FOS Development Facility is stored at the DMS for later use by the User Interface Subsystem.

Flight Dynamics Facility - The DMS receives orbital information from the Flight Dynamics Facility. The orbital information is validated and stored in data files and database tables.

3.2 PDB Ingest

3.2.1 PDB Ingest Context

The PDB Context diagram represents the interface overview of the FOS Database. Definitions are received from external sources to the Data Management Subsystem, processed within, and made available for operational use to other FOS Subsystems.

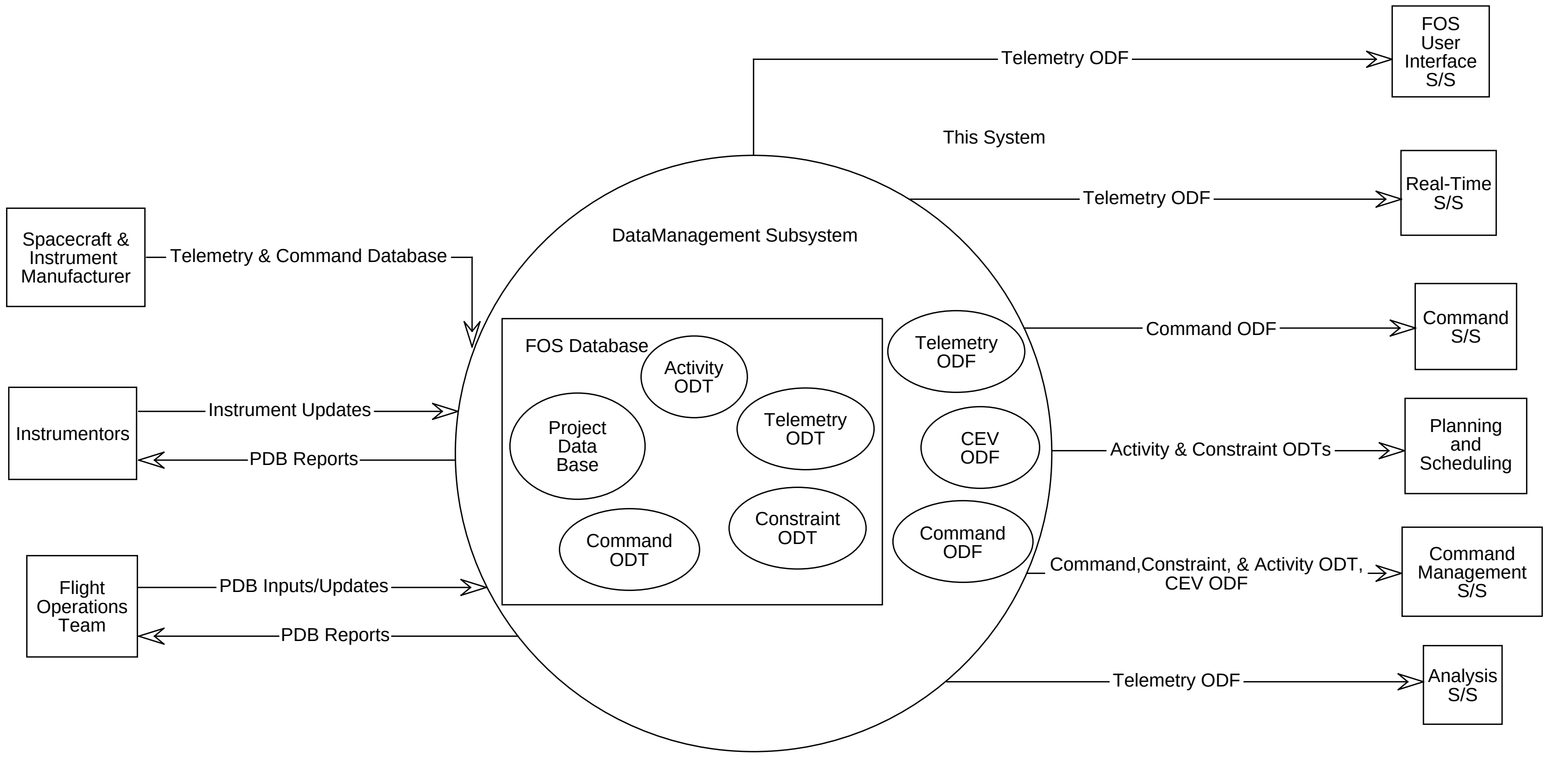


Figure 3.2-1. PDB Ingest Context

3.2.2 PDB Ingest Interfaces

Table 3.2-1. PDB Ingest Interfaces

Interface Service	Interface Class	Interface Class Description	Service Provider	Service User	Frequency
Invoke PDB Database Initialization	FdDbFuiInterface	Provide interface screens to invoke PDB Database Initialization	FUI	DMS	upon delivery of IT database
Invoke PDB Ingest	FdDbFuiInterface	Provide interface screens to invoke PDB Ingest	FUI	DMS	as needed
Invoke PDB Edits	FdDbFuiInterface	Provide interface screens to invoke PDB Edits	FUI	DMS	as needed
Invoke PDB Reporting	FdDbFuiInterface	Provide interface screens to invoke PDB Reporting	FUI	DMS	after PDB validation as needed

3.2.3 PDB Ingest Object Model

The base class FdDbPDBInput represents the input definitions to the EOS AM-1 Project Database (PDB). It consists of the subclasses FdDbTelemetryDefs, FdDbCommandDefs, FdDbConstraintDefs, FdDbActivityDefs. Additionally, the input definitions are provided by the Integration & Test Database, the FOT and as updates from the Instrument Operations Teams.

FdDbProjectDatabase represent the EOS AM-1 Project Database (PDB) that resides at the EOC. This collection of telemetry, command, constraint and activity definitions are derived from the base class FdDbProjectDatabase and are presented in the subclasses FdDbTelemetryPDB, FdDbCommandPDB, FdDbConstraintPDB, FdDbActivityPDB, respectively.

The FdDbLoadPDBInput class is responsible for controlling the loading of the PDB input definitions into the PDB structure at the EOC. Upon completion of this process, the PDB resides as the FdDbUnvalProjectDatabase class where it awaits validation.

3.2.4. PDB Ingest Dynamic Model

3.2.4.1 PDB Ingest Scenario Abstract

The PDB Ingest scenario describes the process of loading the definitions files into the PDB database table structures at the EOC.

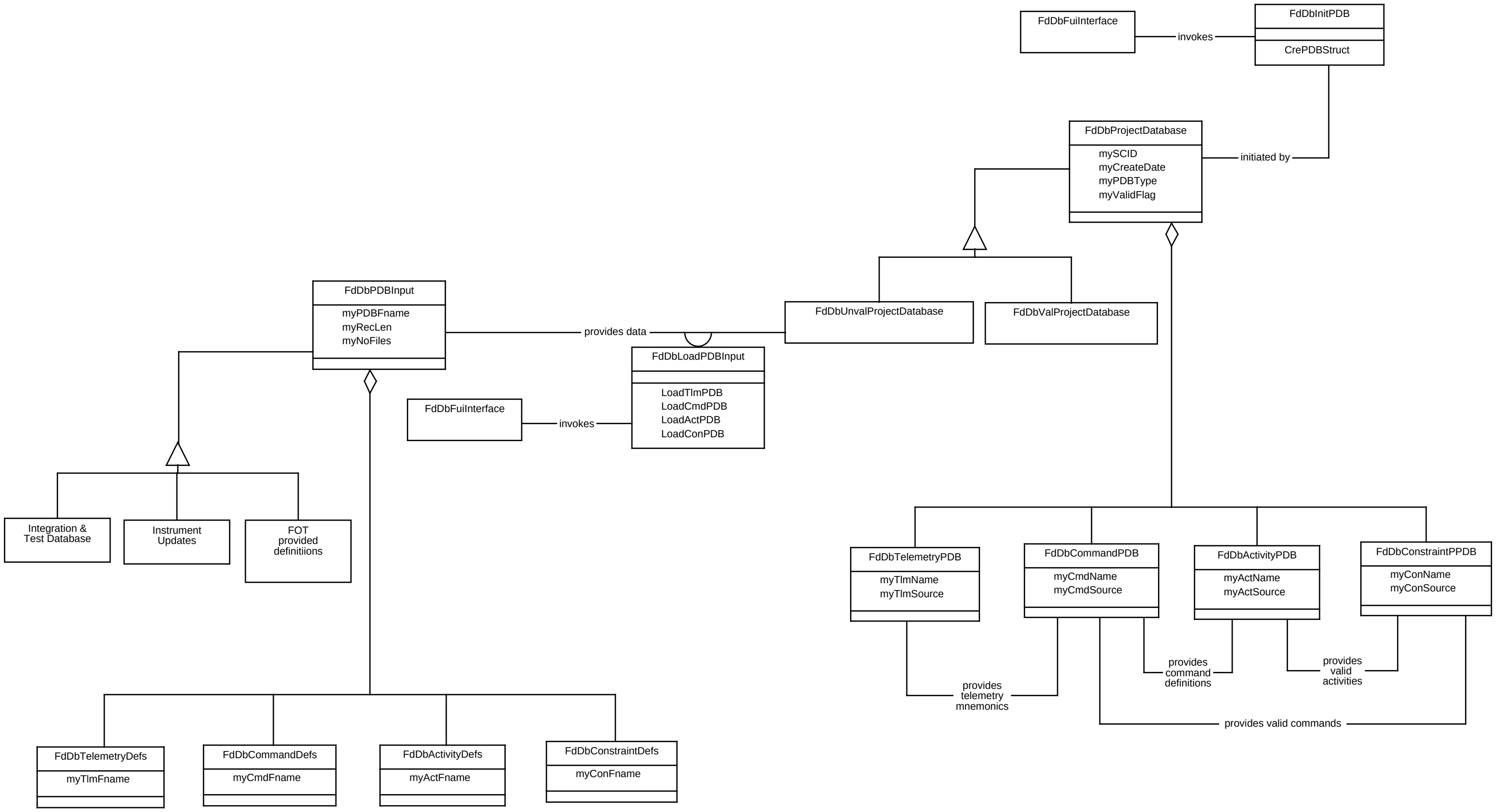


Figure 3.2-2. PDB Ingest Object Model

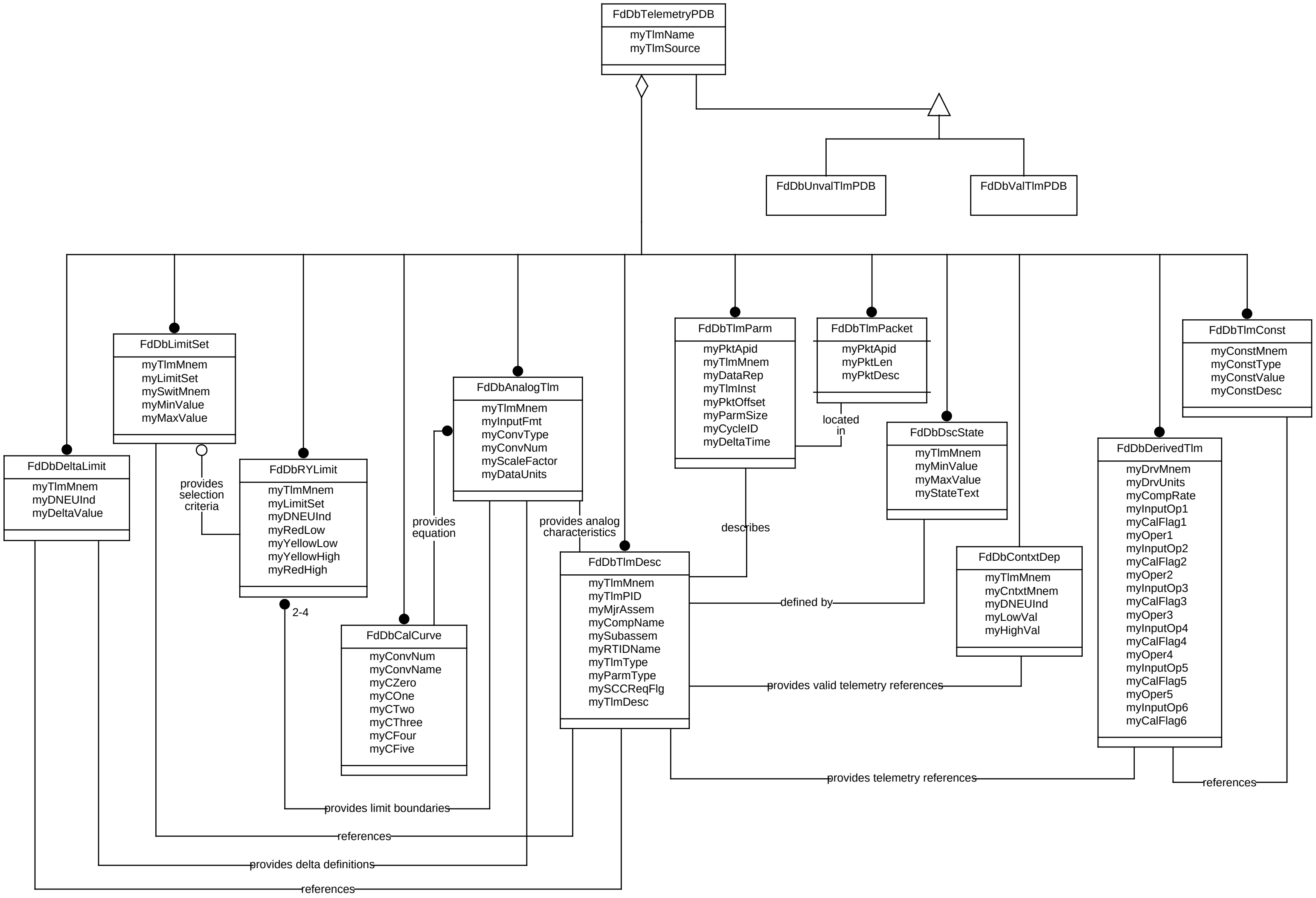


Figure 3.2-3. PDB Ingest Object Model

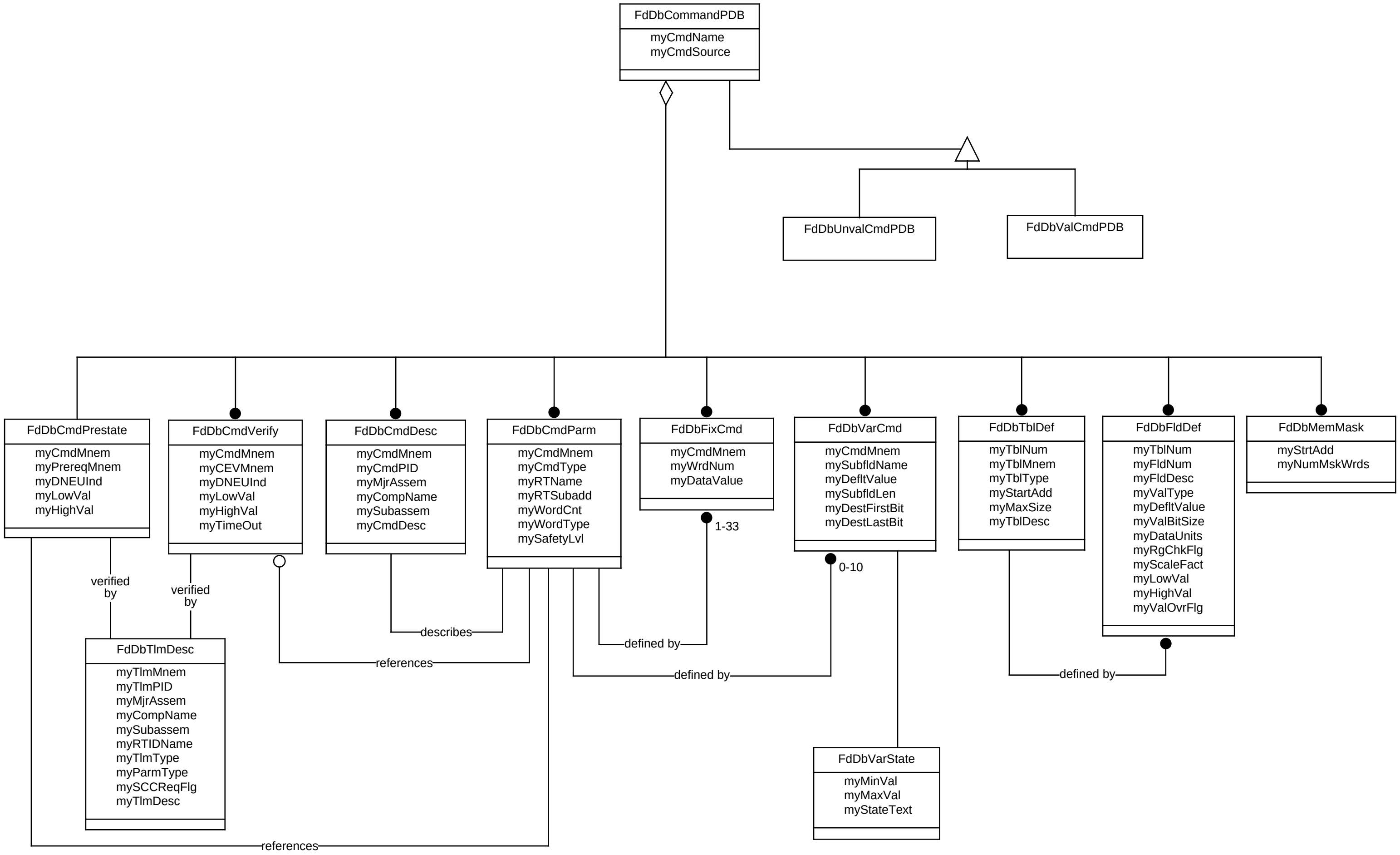


Figure 3.2-4. PDB Ingest Object Model

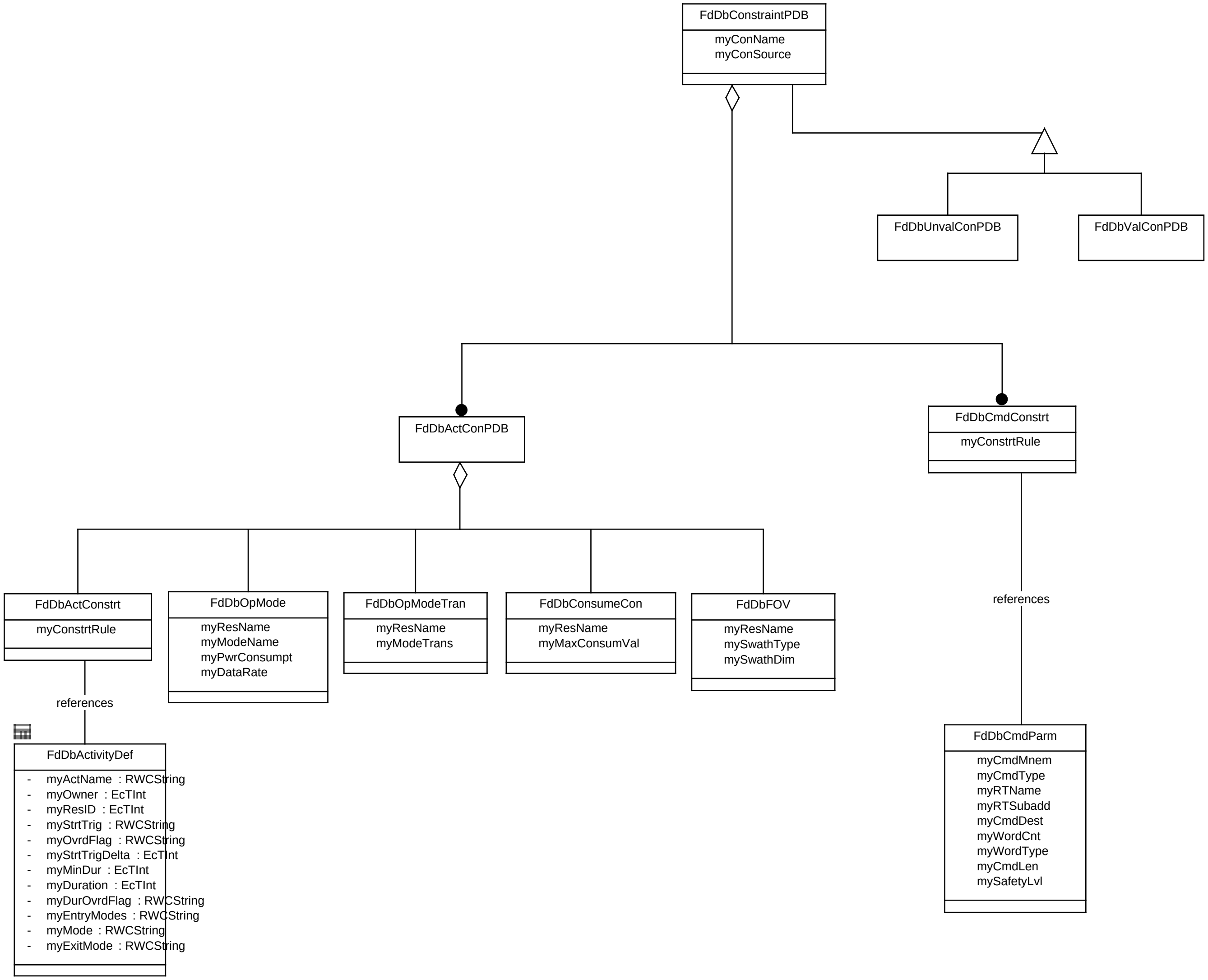


Figure 3.2-5. PDB Ingest Object Model

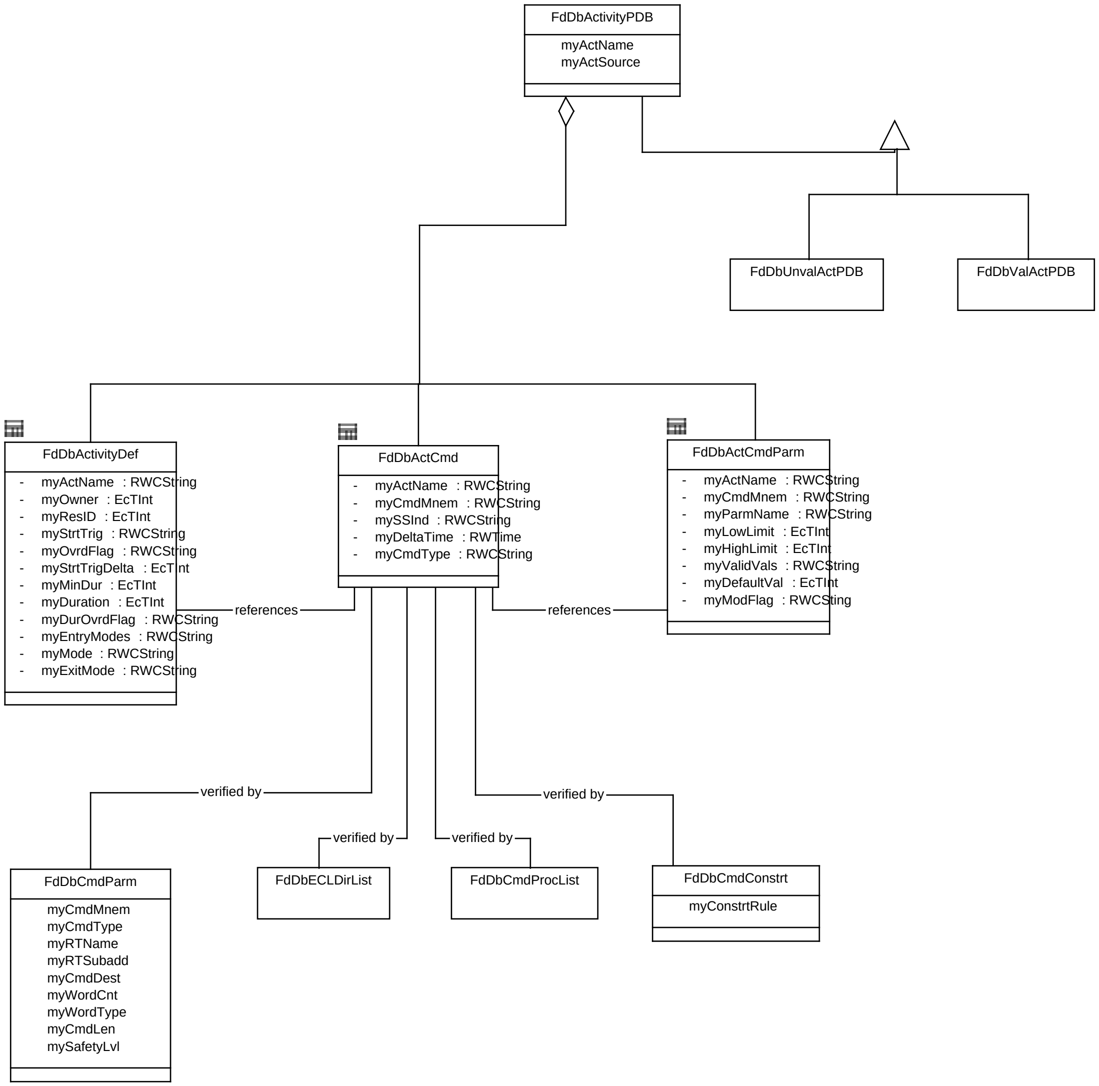


Figure 3.2-6. PDB Ingest Object Model

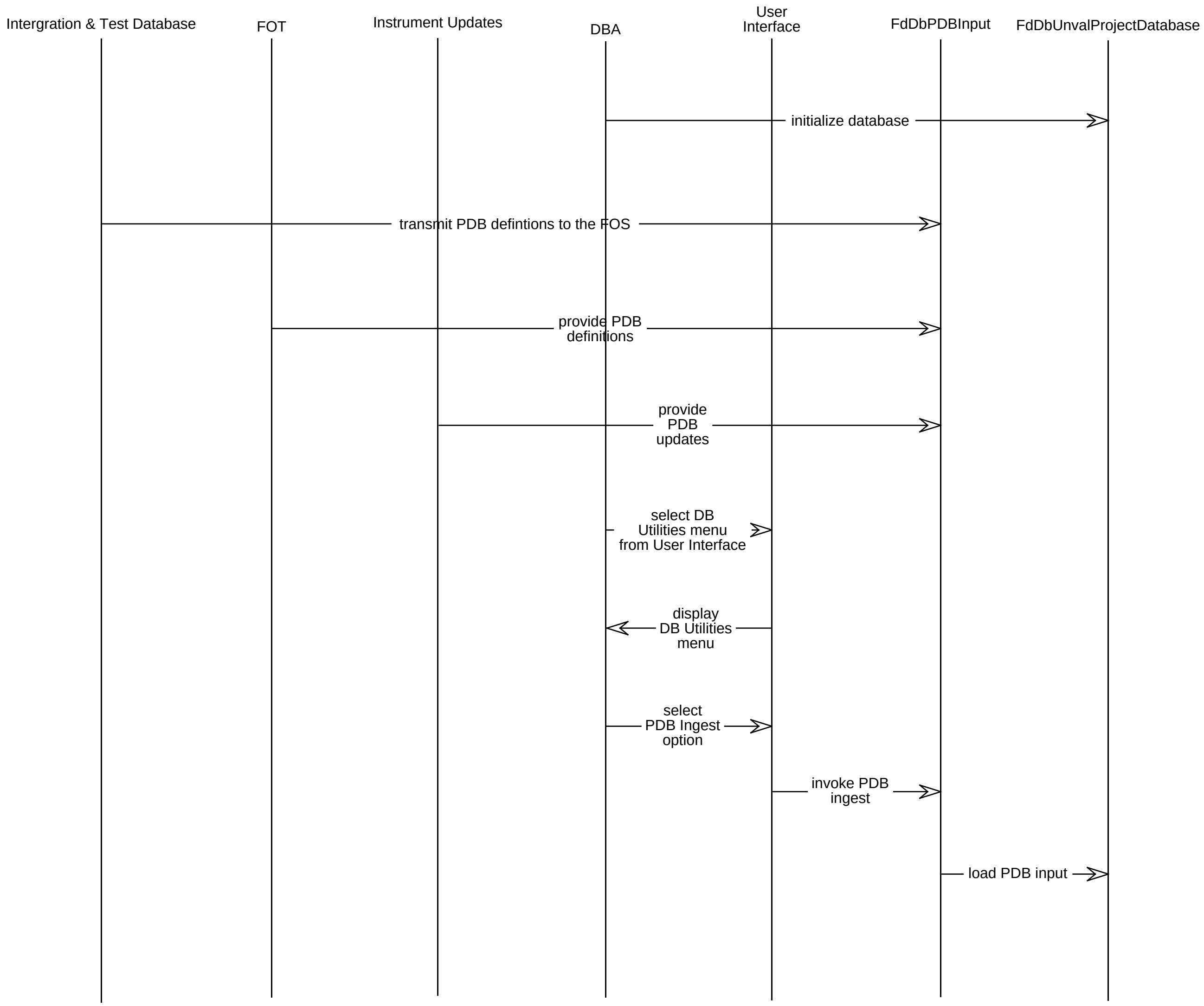


Figure 3.2-7. PDB Ingest Event Trace

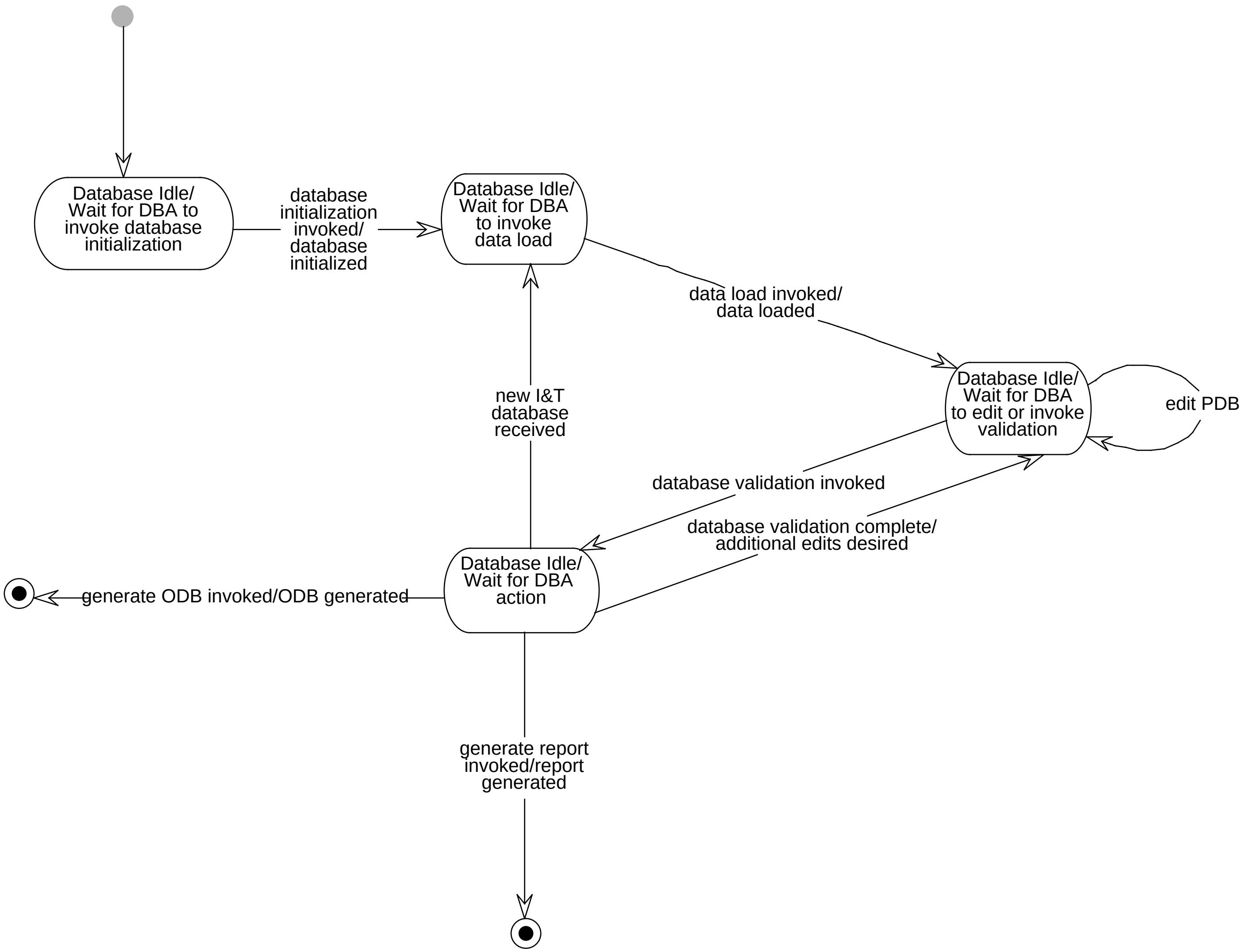


Figure 3.2-8. PDB Ingest State Diagram

3.2.4.2 PDB Ingest Summary Information

Interfaces:

User Interface

Stimulus:

DBA selection of the PDB ingest option

Desired Response:

The loading of the telemetry, command, constraint and activity definitions into the PDB database table structures.

Pre-Conditions:

Database up and running.

Database table structures have been initialized.

Definitions files have been transferred to a dedicated directory at the EOC.

Post-Conditions:

The PDB definitions have been loaded into the internal database structures.

3.2.4.3 PDB Ingest Scenario Description

PDB ingest is an operational function invoked by the Database Administrator (DBA). The selection of the PDB ingest option from the Database Utilities menu begins the process of loading the definitions files into the database table structure at the EOC. Upon completion, the PDB is ready for validation.

3.2.5 PDB Ingest Data Dictionary

Note: Refer to the DFCD for the EOS AM-1 PDB and the FOS Database Design and Database Schema Specifications for specific details supporting the design of PDB processing.

Class Name: FdDbActConPDB

The Activity Constraint PDB class represents the activity-level constraints that are defined for instruments, spacecraft subsystems and ground system components.

Class Name: FdDbActConstrt

The Activity Constraint class represents the activity-level constraints rules.

Class Name: FdDbActCmd

The Activity Command class provides the definitions of commands that make up a specific activity.

Class Name: FdDbActCmdParm

The Activity Command Parameter class provides the definitions of the parameters for each commands that makes up a specific activity.

Class Name: FdDbActivityDef

The Activity Definition class provides the attributes of an activity.

Class Name: FdDbActivityPDB

The Activity PDB represents the activity definition files used to support FOS operations.

Class Name: FdDbAnalogTlm

The Analog Telemetry class provides characteristic information about analog telemetry parameters.

Class Name: FdDbCalCurve

The Calibration Curve class defines the coefficients used to convert raw telemetry values into EUs. Each polynomial calibration equation may specify up to 6 coefficients (e.g., 5th order polynomial). At a minimum, each equation must contain 2 coefficients.

Class Name: FdDbCmdConstrt

The Command Constraint class indicates the command-level constraints that are defined for instruments, spacecraft subsystems and ground system components.

Class Name: FdDdCmdDesc

The Command Description class provides descriptive information about a spacecraft or instrument command parameter.

Class Name: FDbCmdParm

The Command Parameter class defines a spacecraft or instrument command which is used to support the EOS AM-1 spacecraft.

Class Name: FdDbCmdVerify

The Command Execution Verification (CEV) class defines telemetry parameters used to verify the reception and execution of an associated command by the spacecraft subsystem or instrument.

Class Name: FdDbCommandPDB

The Command PDB class represents the command definitions files needed to support commanding of the EOS AM-1 spacecraft.

Class Name: FdDbConstraintPDB

The Constraint PDB class represents the constraint definition files needed to support constraint checking for commands and activities during FOS operations.

Class Name: FdDbConsumeCon

The Consumable Constraint class represents a modeling parameter that can be consumed and replenished.

Class Name: FdDbDeltaLimit

The Delta Limit class defines delta limit checking criteria associated with an analog telemetry parameter.

Class Name: FdDbDerivedTlm

The Derived Telemetry class defines simple equations that combine previously defined analogs, discretes, constants and other derived parameters via arithmetic or logical functions.

Class Name: FdDbDscState

The Discrete States class associates a single text state to a range of values for a discrete telemetry parameter.

Class Name: FdDbFixCmd

The Fixed Data Word Specification class defines the optional data words associated with a command.

Class Name: FDbFldDef

The Table Field Definition class defines entries within the spacecraft or instrument table.

Class Name: FdDbFOV

The Field-Of-View Specification class identifies the shape and dimensions associated with an instrument or spacecraft subsystem sensor swath.

Class Name: FdDbLimitSet

The Limit Selection Specification class defines the selection criteria for setting telemetry parameter limits.

Class Name: FDbMemMask

The Memory Masking Definition class identifies an area of spacecraft instrument memory which is ignored when comparing the dump and ground memory image.

Class Name: FdDbOpMode

The operational mode identifies an operational state associated with an instrument, spacecraft subsystem or EOC ground system component.

Class Name: FdDbOpModeTran

The Operational Mode Specification class indicates the valid operational state transitions for instrument, spacecraft subsystems or ground system components as defined at the level mode.

Class Name: FdDbProjectDatabase

The Project Database class represents the telemetry, command, constraint and activity definition files needed to support FOS operations.

Class Name: FdDbRYLimit

The Red/Yellow Limit Specification record defines the red/yellow - high/low limit checking criteria associated with an analog or discrete telemetry parameter.

Class Name: FdDbTblDef

The Table Definition class defines area of the spacecraft or instrument memory.

Class Name: FdDbTelemetryPDB

The Telemetry PDB class represents the telemetry definition files needed to support telemetry processing during FOS operations.

Class Name: FdDbTlmDesc

The Telemetry Description class provides descriptive information about a telemetry parameter.

Class Name: FdDbTlmPacket

The Telemetry Packet Specification class defines valid CCSDS packets for processing by the FOS.

Class Name: FdDbTlmParm

The Telemetry Parameter Specification class provides the mapping tables used to decommutate the downlink telemetry streams into specific analog or discrete telemetry mnemonics.

Class Name: FDbVarCmd

The Command Variable Data Word Specification class defines the subfields associated with variable type commands.

Class Name: FdDbVarStates

The Variable States class provides the states associated with a subfield.

3.3 PDB Validation

3.3.1 PDB Validation Context

Refer to Section 3.2.1

3.3.2 PDB Validation Interfaces

Refer to Section 3.2.2

3.3.3 PDB Validation Object Model

The FdDbProjectDatabase class represent the AM-1 Project Database (PDB). This collection of definitions files is stored at the EOC, validated and made available for operational use. These files are made up of telemetry, command, constraint and activity definitions (FdDbTelemetryPDB, FdDbCommandPDB, FdDbConstraintPDB, FdDbActivityPDB). The PDB Validation Object Model reflects the process from which the PDB is taken from the class, FdDbUnvalProjectDatabase, to the class, FdDbValProjectDatabase. The FdDbUnvalProjectDatabase class represents the PDB when it has been loaded into the internal PDB structures at the EOC. The FdDbValProjectDatabase class represents the definition files once they have been validated.

Each of the PDB validation subclasses (FdDbValidateTlm, FdDbValidateCmd, FdDbValidateCon, FdDbValidateAct) is derived from the FdDbValidatePDB base class. They are responsible for controlling the validation of each type of PDB definition.

The FdDbValSumLog class is responsible for maintaining errors found during the validation process.

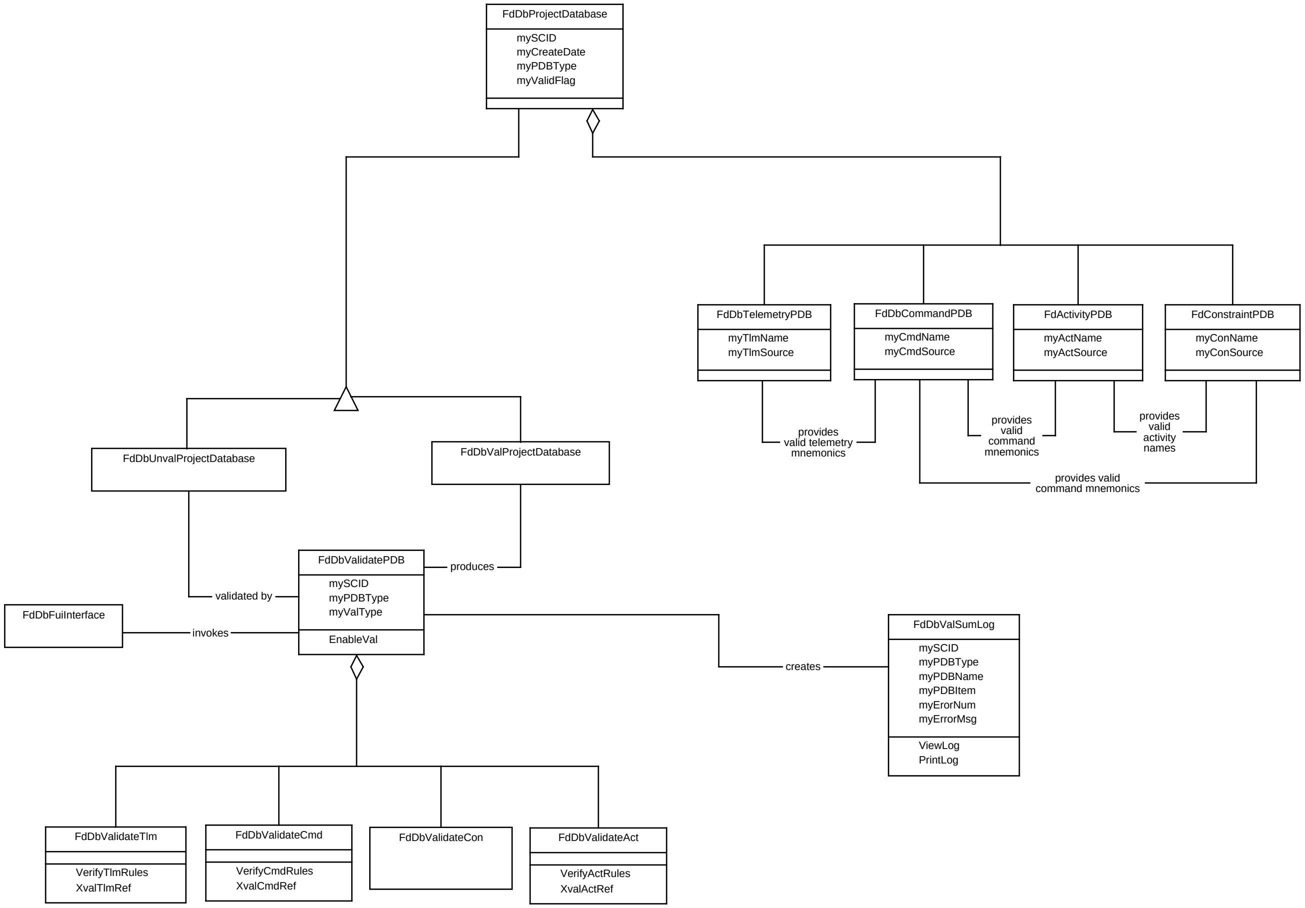


Figure 3.3-1. PDB Validation Object Model

3.3.4. PDB Validation Dynamic Model

3.3.4.1 PDB Validation Scenario Abstract

The PDB validation scenario describes the process in which the definitions files used to support FOS operations are validated.

3.3.4.2 PDB Validation Summary Information

Interfaces:

User Interface

Stimulus:

DBA selection of PDB validation

Desired Response:

The creation of the validated telemetry, command, constraint and activity PDB.

Creation and generation of a PDB validation summary log.

Pre-Conditions:

Database up and running.

Database table structures have been initialized.

PDB definitions have loaded into the internal database table structures.

Post-Conditions:

Validated PDB

3.3.4.3 PDB Validation Scenario Description

PDB validation is an operational function invoked by the Database Administrator (DBA). Through the selection of the PDB validation option on the Database Utilities menu, this process begins with the validation of the telemetry definitions. PDB validation is ordered by the PDB type to ensure the integrity of the definitions. Next, the command definitions are validated, followed by the validation of the constraint and activities definitions. The telemetry and command definitions are provided by the AM-1 integration and test database. Each time changes occur to the telemetry and command definitions maintained at the EOC, validation of the entire PDB is required. Constraint and activity definitions are provided by the FOT through the use of database interface tools. The changes to this data occur independent of the telemetry and command definition changes. For this reason, the constraint and activity PDB may also be validated when only their changes occur. Validation errors are reported in a validation summary log.

3.3.5 PDB Validation Data Dictionary

Reference Section 3.2.5 PDB Ingest Data Dictionary.

Note: Refer to the DFCD for the EOS AM-1 PDB and the FOS Database Design and Database Schema Specifications for specific details supporting the design of PDB processing.

3.4 PDB Edit

3.4.1 PDB Edit Context

Refer to Section 3.2.1.

3.4.2 PDB Edit Interfaces

Refer to Section 3.2.2.

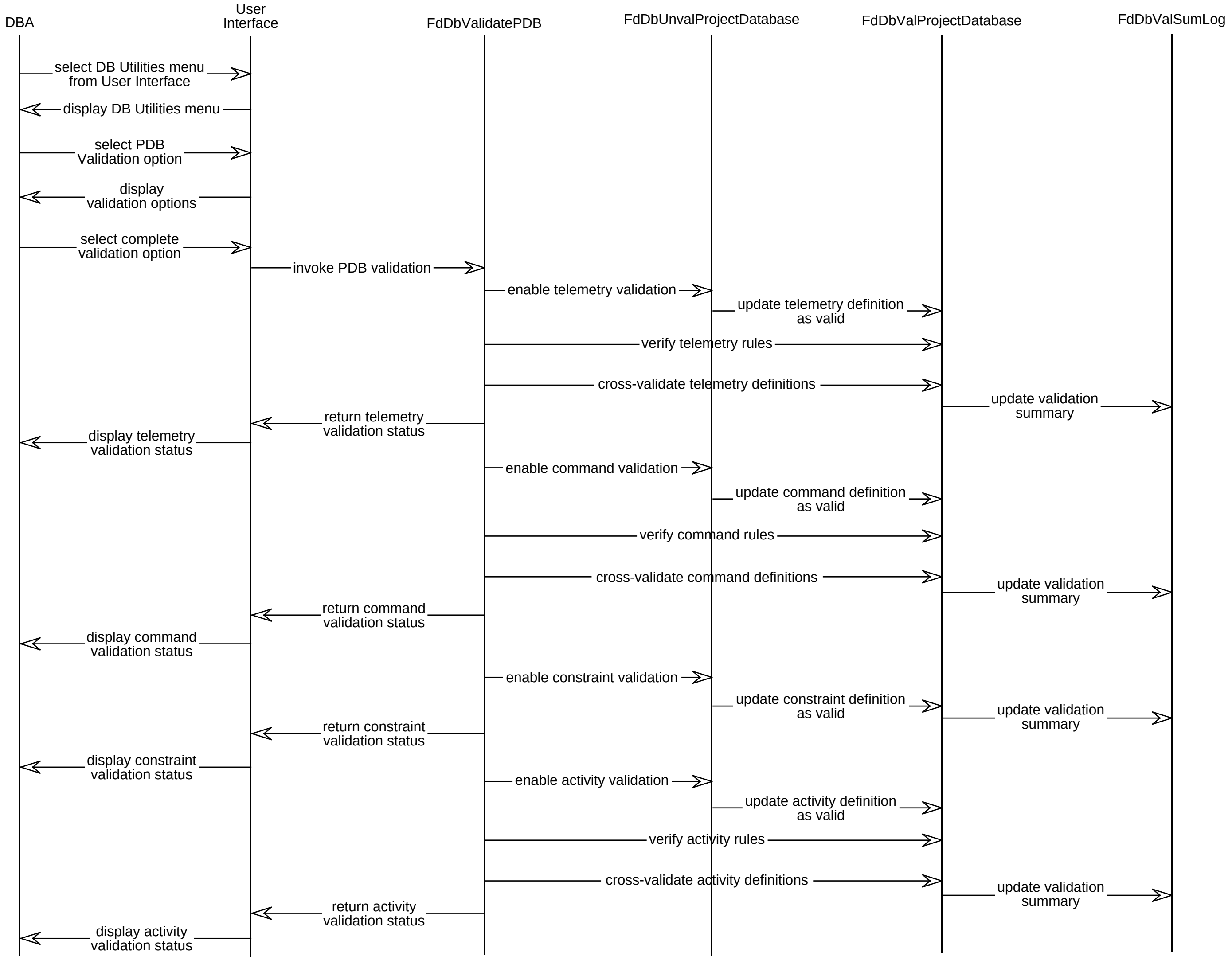

DBA
User Interface
FdDbValidatePDB
FdDbUnvalProjectDatabase
FdDbValProjectDatabase
FdDbValSumLog
select DB Utilities menu from User Interface
display DB Utilities menu
select PDB Validation option
display validation options
select complete validation option
invoke PDB validation
enable telemetry validation
update telemetry definition as valid
verify telemetry rules
cross-validate telemetry definitions
update validation summary
return telemetry validation status
display telemetry validation status
enable command validation
update command definition as valid
verify command rules
cross-validate command definitions
update validation summary
return command validation status
display command validation status
enable constraint validation
update constraint definition as valid
update validation summary
return constraint validation status
display constraint validation status
enable activity validation
update activity definition as valid
verify activity rules
cross-validate activity definitions
update validation summary
return activity validation status
display activity validation status

Figure 3.3-2. PDB Validation Event Trace

3.4.3 PDB Edit Object Model

FdDbEditPDB represents the database editor interface class to perform edits to the AM-1 Project Database (PDB). The FdDbUnvalProjectDatabase class provides data to the FdDbEditPDB class. (The FdDbUnvalProjectDatabase class is derived from the FdDbProjectDatabase class and is described in Section 3.2.) The FdDbEditPDB class is made up of the FdDbEditTlmScrn, FdDbEditCmdScrn, FdDbEditActScrn, and FdDbEditConScrn subclasses. The FdDbEditPDB class provides the capability to retrieve data, delete data, save data, and move between data records in the case of a multi-record retrieval. All edits made to the PDB are logged by the FdDbEditLog class. This class provides the capability to send the log to the printer or view the log from the screen.

3.4.4 PDB Edit Dynamic Model

3.4.4.1 PDB Edit Scenario Abstract

3.4.4.2 PDB Edit Summary Information

Interfaces:

User Interface

Stimulus:

DBA selection of the PDB Edit option

Desired Response:

Edits to the unvalidated PDB

Pre-Conditions:

The database is up and running.

The user has privileges to edit data.

Post-Conditions:

Modified data is stored in the database.

3.4.4.3 PDB Edit Scenario Description

The Project Data Base (PDB) Edit process is initiated through the selection of the Project Data Base (PDB) Edit option on the Database Utilities Menu by a user authorized to make edits to the unvalidated PDB.

The user specifies which PDB data to edit through User Interface prompts. Once the user has made data type selections, the database edit screen is invoked with the data fields displayed. (This screen was developed using a database manipulation COTS product.)

The database editor screen consists of data fields and database manipulation buttons. The buttons are used to query data from the database, manipulate records retrieved from the database, clear data from the data fields, and save new or modified data to the database tables. Edit messages are displayed on the bottom of the screen.

The editor can be exited by selecting an exit button on the screen.

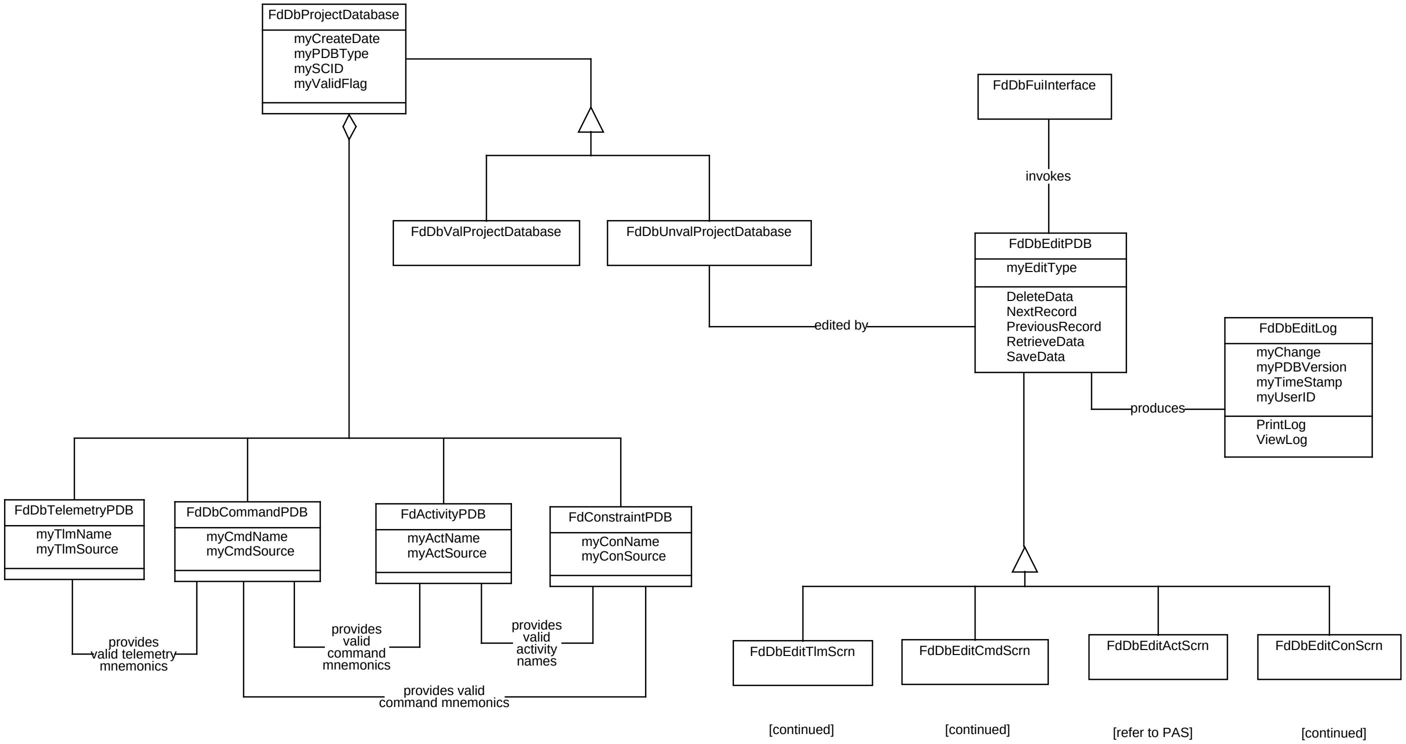


Figure 3.4-1. PDB Edit Object Model

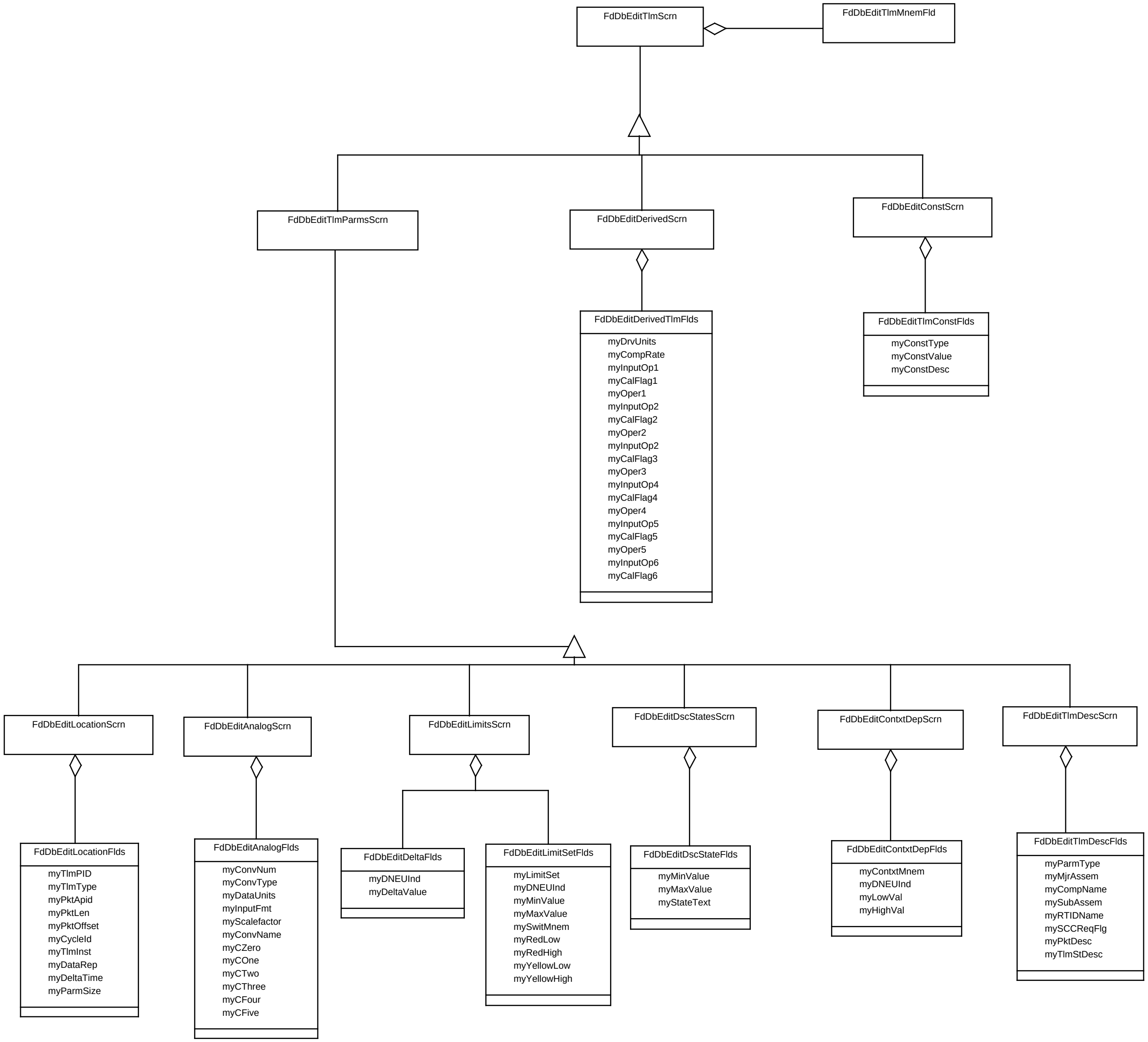


Figure 3.4-2. PDB Edit Object Model

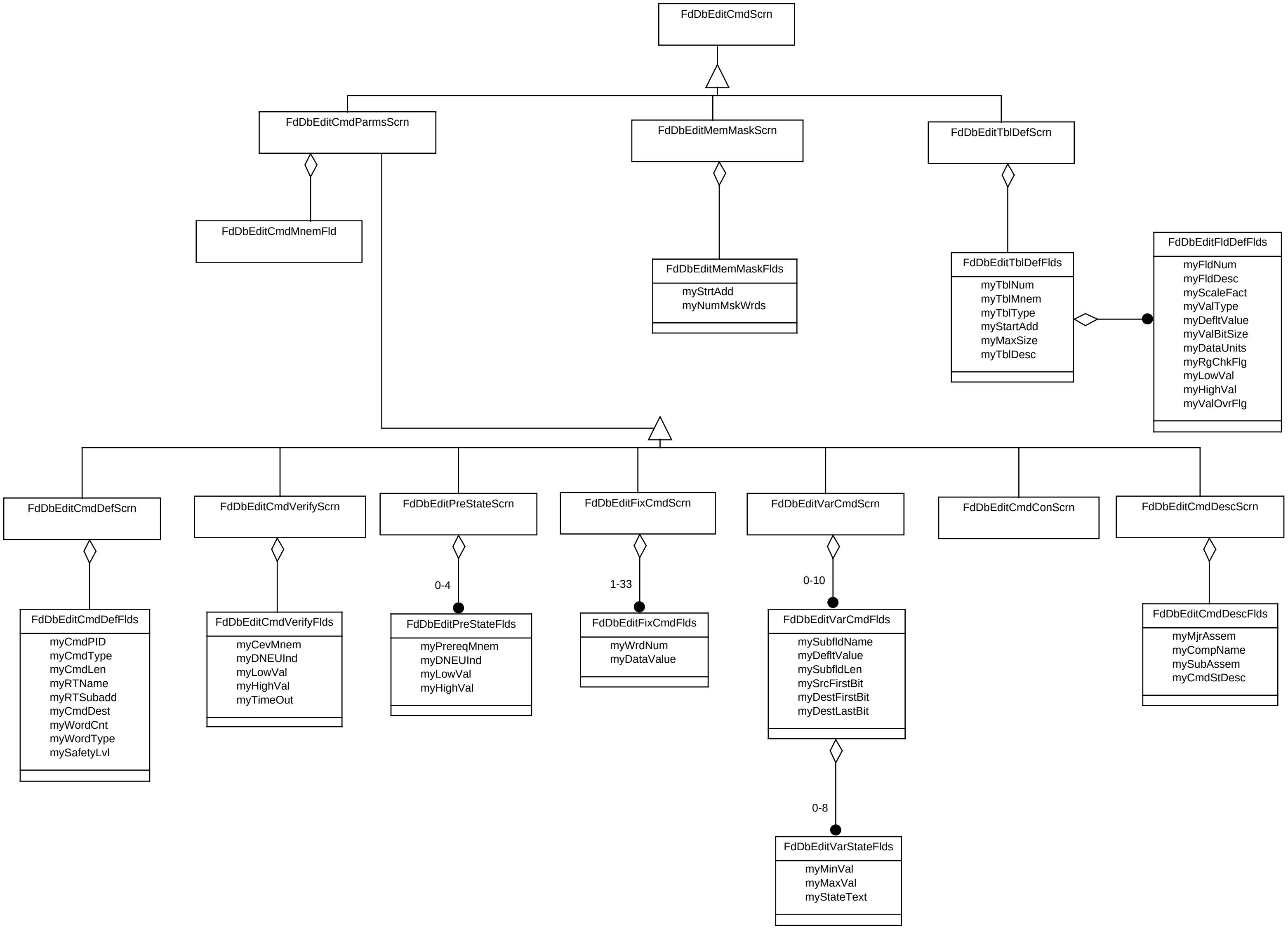


Figure 3.4-3. PDB Edit Object Model

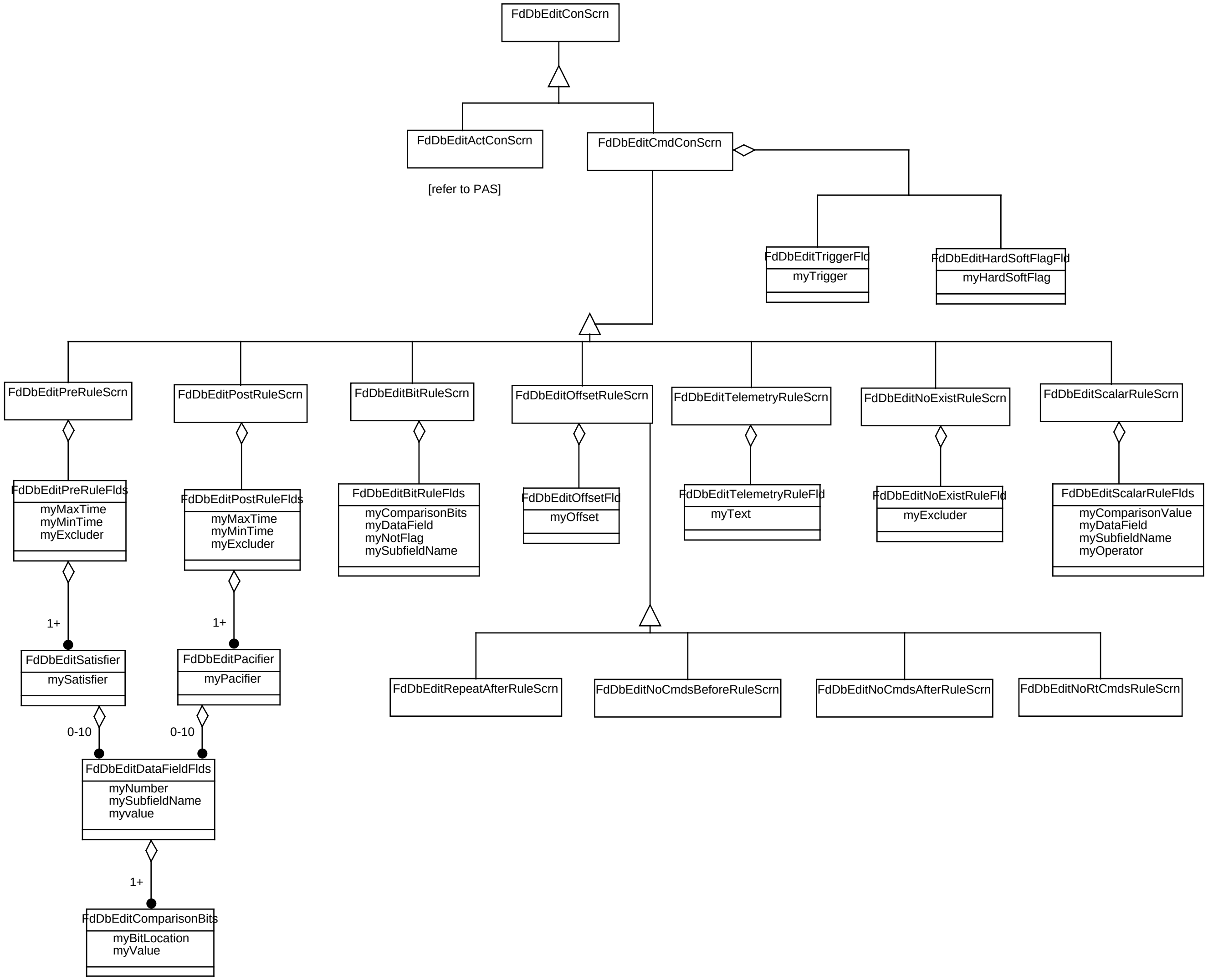


Figure 3.4-4. PDB Edit Object Model

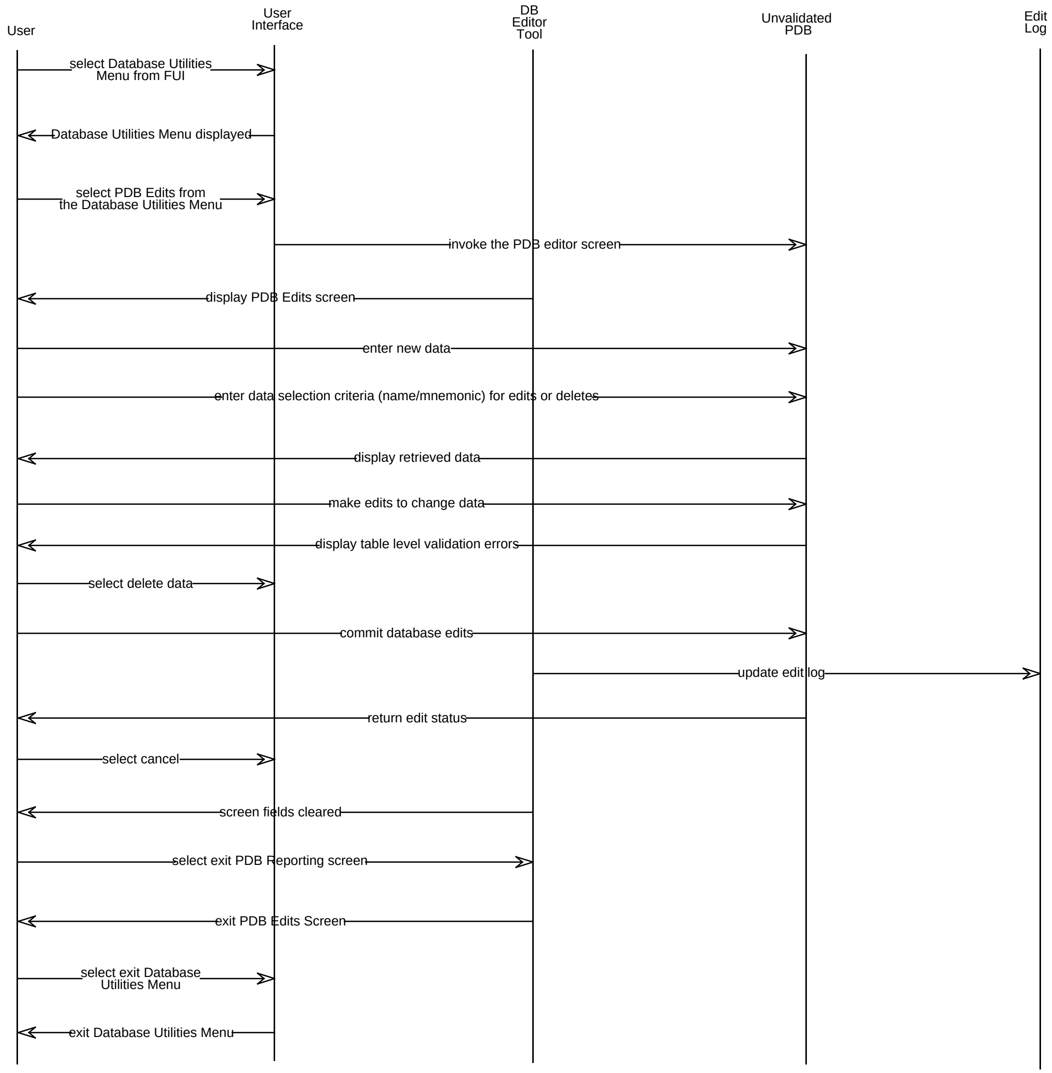

User
User Interface
DB Editor Tool
Unvalidated PDB
Edit Log
select Database Utilities Menu from FUI
Database Utilities Menu displayed
select PDB Edits from the Database Utilities Menu
invoke the PDB editor screen
display PDB Edits screen
enter new data
enter data selection criteria (name/mnemonic) for edits or deletes
display retrieved data
make edits to change data
display table level validation errors
select delete data
commit database edits
update edit log
return edit status
select cancel
screen fields cleared
select exit PDB Reporting screen
exit PDB Edits Screen
select exit Database Utilities Menu
exit Database Utilities Menu

Figure 3.4-5. PDB Edit Event Trace

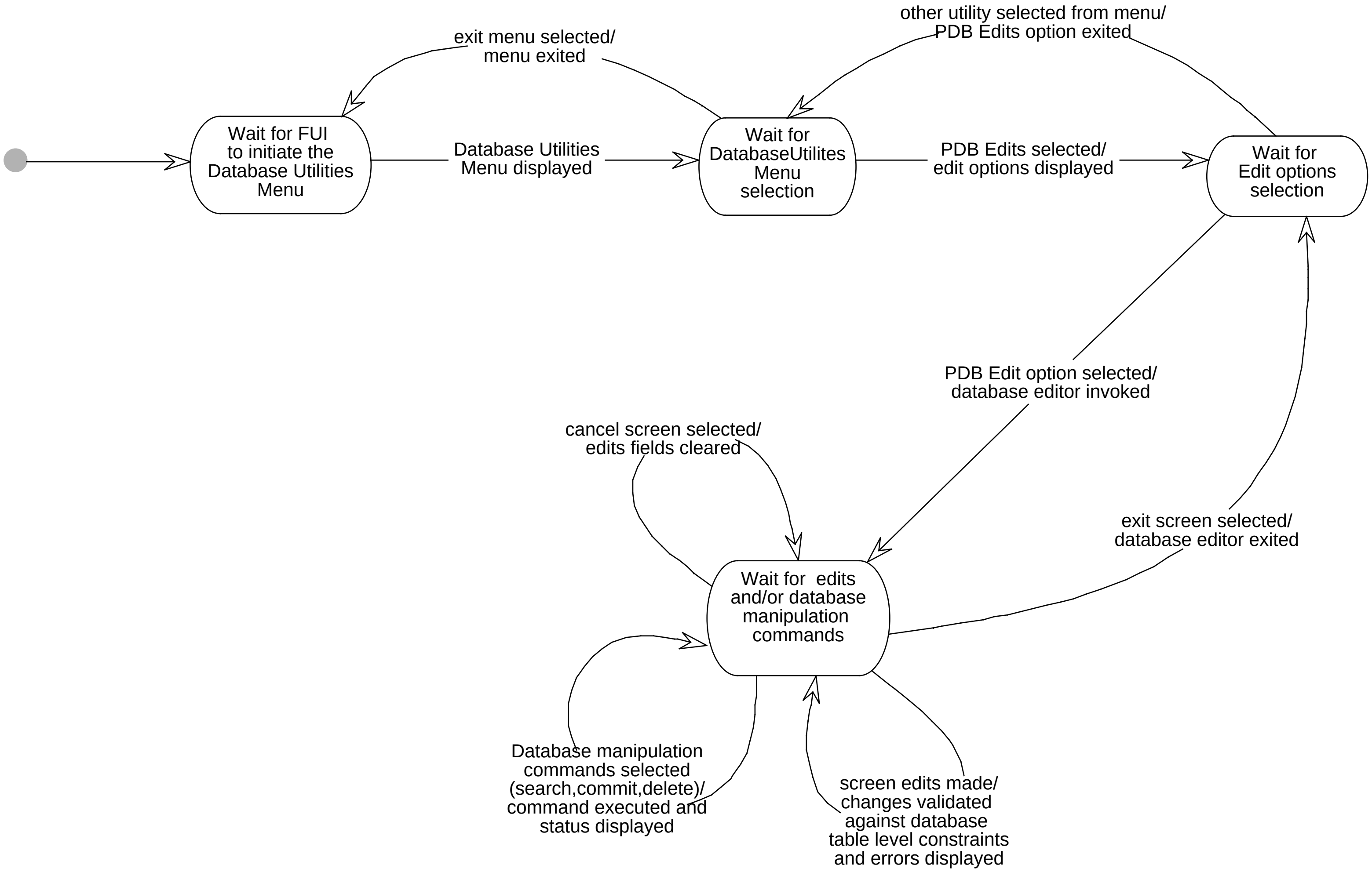


Figure 3.4-6. PDB Edit State Diagram

3.4.5 PDB Edit Data Dictionary

Note: Refer to the DFCD for the EOS AM-1 PDB and the FOS Database Design and Database Schema Specifications for specific details supporting the design of PDB processing.

Class Name: FdDbEditActScrn

Description: The Edit Activity Screen class provides the user interface window for editing the activity PDB.

Class Name: FdDbEditAnalogFlds

Description: The Edit Analog Fields class provides the fields associated with the telemetry parameter analog definitions for editing.

Class Name: FdDbEditAnalogScrn

Description: The Edit Analog Screen class provides the user interface window for editing the telemetry parameter analog definitions.

Class Name: FdDbEditBitRuleFlds

Description: The Edit Bit Rule Fields class provides the fields associated with the command constraint bit rule definitions for editing.

Class Name: FdDbEditBitRuleScrn

Description: The Edit Bit Rule Screen class provides the user interface window for editing the command constraint bit rule definitions.

Class Name: FdDbEditCmdParmsScrn

Description: The Edit Command Parameters Screen class provides the user interface window for editing the command parameter definitions.

Class Name: FdDbEditCmdDescFlds

Description: The Edit Command Description Fields class provides the fields associated with the command parameter description definitions for editing.

Class Name: FdDbEditCmdDescScrn

Description: The Edit Command Description Screen class provides the user interface window for editing the command parameter description definitions.

Class Name: FdDbEditCmdConfFlds

Description: The Edit Command Constraint Fields class provides the fields associated with the command parameter constraints for editing.

Class Name: FdDbEditCmdConScrn

Description: The Edit Command Constraints Screen class provides the user interface window for editing the command parameter constraint definitions.

Class Name: FdDbEditCmdDefFlds

Description: The Edit Command Definition Fields class provides the fields associated with the command parameter definitions for editing.

Class Name: FdDbEditCmdDefScrn

Description: The Edit Command Definitions Screen class provides the user interface window for editing the command parameter definitions.

Class Name: FdDbEditCmdDescFlds

Description: The Edit Command Description Fields class provides the fields associated with the command parameter description definitions for editing.

Class Name: FdDbEditCmdDescScrn

Description: The Edit Command Description Screen class provides the user interface window for editing the command parameter description defintions.

Class Name: FdDbEditCmdMnemFld

Description: The Edit Command Mnemonic Field class provides the field associated with the command mnemonic for editing.

Class Name: FdDbEditCmdScrn

Description: The Edit Command Screen class provides the user interface window for editing the command PDB.

Class Name: FdDbEditCmdVerifyFlds

Description: The Edit Command Verify Fields class provides the fields associated with the command parameter verify definitions for editing.

Class Name: FdDbEditCmdVerifyScrn

Description: The Edit Command Verify Screen class provides the user interface window for editing the command parameter verify definitions.

Class Name: FdDbEditComparisonBitsFlds

Description: The Edit Comparison Bits Fields class provides the fields associated with the command constraint comparison bits definition associated with a symbol defintion for editing.

Class Name: FdDbEditConScrn

Description: The Edit Constraint Screen class provides the user interface window for editing the constraint PDB.

Class Name: FdDbEditContxtDepFlds

Description: The Edit Context Dependent Fields class provides the fields associated with the telemetry parameter context dependent definitions for editing.

Class Name: FdDbEditContxtDepScrn

Description: The Edit Context Dependent Screen class provides the user interface window for editing the telemetry parameter context dependent definitions.

Class Name: FdDbEditDataFieldFlds

Description: The Edit Data Field Fields class provides the fields associated with the command constraint data fields definition associated with a pre rule or a post rule for editing.

Class Name: FdDbEditDeltaFlds

Description: The Edit Delta Limit Fields class provides the fields associated with the telemetry parameter delta limit definitions for editing.

Class Name: FdDbEditDerivedTlmFlds

Description: The Edit Derived Telemetry Fields class provides the fields associated with the derived telemetry definitions for editing.

Class Name: FdDbEditDerivedTlmScrn

Description: The Edit Derived Screen class provides the user interface window for editing the derived telemetry definitions.

Class Name: FdDbEditDscStateFlds

Description: The Edit Discrete State Fields class provides the fields associated with the telemetry parameter discrete state definitions for editing.

Class Name: FdDbEditDscStatesScrn

Description: The Edit Discrete States Screen class provides the user interface window for editing the telemetry parameter discrete state definitions.

Class Name: FdDbEditFixCmdFlds

Description: The Edit Fixed Command Fields class provides the fields associated with the command parameter fixed command definitions for editing.

Class Name: FdDbEditFixCmdScrn

Description: The Edit Fixed Command Screen class provides the user interface window for editing the command parameter fixed state definitions.

Class Name: FdDbEditHardSoftFlagFld

Description: The Edit Hard Soft Flag Field class provides the field associated with the command constraint hard/soft flag for editing.

Attributes:

myHardSoftFlag: string

Description: hard/soft flag identifies the hard/soft flag defined for the specifiec command constraint

Class Name: FdDbEditLimitSetFlds

Description: The Edit Limit Set Fields class provides the fields associated with the telemetry parameter limit set definitions for editing.

Class Name: FdDbEditLimitsScrn

Description: The Edit Limits Screen class provides the user interface window for editing the telemetry parameter limits definitions.

Class Name: FdDbEditLocationFlds

Description: The Edit Location Fields class provides the fields associated with the Telemetry Parameter Location definitions for editing.

Class Name: FdDbEditLocationScrn

Description: The Edit Location Screen class provides the user interface window for editing the Telemetry Parameter Location definitions.

Class Name: FdDbEditMemMaskFlds

Description: The Edit Memory Mask Fields class provides the fields associated with the command memory mask definitions for editing.

Class Name: FdDbEditMemMaskScrn

Description: The Edit Memory Mask Screen class provides the user interface window for editing the telemetry memory mask definitions.

Class Name: FdDbEditPacifier

Description: The Edit Pacifier class provides the field associated with the command post rule constraint pacifier for editing.

Class Name: FdDbNoCmdsAfterRuleFlds

Description: The Edit No Commands After Rule Screen class provides the user interface window for editing the command constraint no commands after rule offset rule.

Class Name: FdDbNoCmdsBeforeRuleFlds

Description: The Edit No Commands Before Rule Screen class provides the user interface window for editing the command constraint no commands before rule offset rule.

Class Name: FdDbEditNoExistRuleFld

Description: The Edit No Exist Rule Fld class provides the field associated with the command constraint no exist rule definition for editing.

Class Name: FdDbEditNoExistRuleScrn

Description: The Edit No Exist Rule Screen class provides the user interface windowfor editing the command constraint no exist rule definitions.

Class Name: FdDbNoRTCmdsRuleScrn

Description: The Edit No Real-time Commands Rule Screen class provides the user interface window for editing the command constraint no real-time commands rule offset rule.

Class Name: FdDbEditOffsetFld

Description: The Edit Offset Field class provides the field associated with the command constraint offset rules for editing.

Class Name: FdDbEditOffsetRuleScrn

Description: The Edit Offset Screen class provides the user interface window for editing the command constraint offset rule definitions.

Class Name: FdDbEditPDB

Description: The Edit PDB class represents the edit screen for editing the PDB data definitions.

Attributes:

myEditType: string

Description:edit type indicates the type of data being edited (telemetry, commmand, activity, constraint)

Operations:

FdDbPDBEdit :: DeleteData

Description:the operation to delete the record associated with the data on the screen

FdDbPDBEdit :: NextRecord

Description:the operation to display the next record in the retrieval buffer

FdDbPDBEdit :: PreviousRecord

Description:the operation to display the previous record in the retrieval buffer

FdDbPDBEdit :: RetrieveData

Description:the operation to retrieve data from the database for the data criteria specified on the screen

FdDbPDBEdit :: SaveData

Description:the operation to save data to the database that has been entered on the screen

Class Name: FdDbEditPreRuleFlds

Description: The Edit Pre Rule Fields class provides the fields associated with the command constraint pre rule definitions for editing.

Class Name: FdDbEditPreRuleScrn

Description: The Edit Pre Rule Screen class provides the user interface window for editing the command constraint pre rule definitions.

Class Name: FdDbEditPostRuleFlds

Description: The Edit Post Rule Fields class provides the fields associated with the command constraint post rule definitions for editing.

Class Name: FdDbEditPostRuleScrn

Description: The Edit Post Rule Screen class provides the user interface window for editing the command constraint post rule definitions.

Class Name: FdDbEditPreStateFlds

Description: The Edit Prerequisite State Fields class provides the fields associated with the command parameter prerequisite state definitions for editing.

Class Name: FdDbEditPreStateScrn

Description: The Edit Prerequisite State Screen class provides user interface window for editing the command parameter prerequisite state definitions.

Class Name: FdDbEditRepeatAfterRuleFlds

Description: The Edit Repeat After Rule Field class provides the field associated with the command constraint repeat after rule definition associated with an offset rule for editing.

Class Name: FdDbEditScalarRuleFlds

Description: The Edit Scalar Rule Fields class provides the fields associated with the command constraint scalar rule definitions for editing.

Class Name: FdDbEditScalarRuleScrn

Description: The Edit Scalar Rule Screen class provides the user interface window for editing the command constraint scalar rule definitions.

Class Name: FdDbEditSatisfier

Description: The Edit Satisfier class provides the field associated with the command pre rule constraint satisfier for editing.

Class Name: FdDbEditTblDefFlds

Description: The Edit Table Definition Fields class provides the fields associated with the command table definitions for editing.

Class Name: FdDbEditTblDefScrn

Description: The Edit Table Definitions Screen class provides the user interface window for editing the telemetry table definitions.

Class Name: FdDbEditTlmConstFlds

Description: The Edit Telemetry Constant Fields class provides the fields associated with the telemetry constants defintions for editing.

Class Name: FdDbEditTlmConstScrn

Description: The Edit Constant Screen class provides the user interface window for editing the telemetry constant definitions.

Class Name: FdDbEditTlmDescFlds

Description: The Edit Telemetry Description Fields class provides the fields associated with the telemetry parameter description definitons for editing.

Class Name: FdDbEditTlmDescScrn

Description: The Edit Telemetry Description Screen class provides the user interface window for editing the telemetry description definitions.

Class Name: FdDbEditTlmParmsScrn

Description: The Edit Telemetry Parmeters Screen class provides the user interface window for editing the telemetry parameter definitions.

Class Name: FdDbEditLog

Description: The Edit Log class represents a record of edits made to the PDB.

Attributes:

myChange: string

Description the type of edit performed on the PDB information.

myPDBVersion: integer

Description the current version of the PDB.

myTimeStamp: string

Description the date and time of the change made to the PDB.

myUserID: string

Description the identification of the user making changes to the PDB.

Operations:

FdDbEditLog::PrintLog

Description: operation to print the edit log.

FdDbEditLog::ViewLog

Description: operation to view the edit log.

Class Name: FdDbEditTlmScrn

Description: The Edit Telelmetry Screen class provides the user interface window for editing the telemetry PDB.

Class Name: FdDbEditTlmParmScrn

Description: The Edit Telelmetry Parameter Screen class provides the user interface windowfor editing the telemetry parameter definitions.

Class Name: FdDbEditTelemetryRuleFld

Description: The Edit Telemetry Field class provides the field associated with the command constraint telemetry rule definitions.

Class Name: FdDbEditTelemetryRuleScrn

Description: The Edit Telemetry Screen class provides the user interface window for editing the command constraint telemetry rule definitions.

Class Name: FdDbEditTlmScrn

Description: The Edit Telemetry Screen class provides the user interface window for editing the telemetry definitions.

Class Name: FdDbEditTriggerFld

Description: The Edit Trigger Field class provides the fields associated with the command trigger for editing command constraint definitions.

Class Name: FdDbEditVarCmdFlds

Description: The Edit Variable Command Fields class provides the fields associated with the command paremater variable command definitions for editing.

Class Name: FdDbEditVarCmdScrn

Description: The Edit Variable Command Screen class provides the user interface window for editing the variable command parameter definitions.

Class Name: FdDbEditVarStateFlds

Description: The Edit Variable State Fields class provides the fields associated with the command parameter variable state definitions for editing.

3.5 PDB Report

3.5.1 PDB Report Context

Refer to Section 3.2.1.

3.5.2 PDB Report Interfaces

Refer to Section 3.2.2.

3.5.3 PDB Report Object Model

FdDbReportPDB represents the database reporting interface class to generate, view, or print reports on the AM-1 Project Database (PDB). The FdDbValProjectDatabase provides data to the FdDbReportPDB class. (The FdDbValProjectDatabase class is derived from the FdDbProjectDatabase class and is described in Section 3.2.) The FdDbReportPDB class is made up of the FdDbTlmRpt, FdDbCmdRpt, FdDbActRpt, and FdDbConRpt subclasses. The FdDbReportPDB class provides the capability to view or print existing reports, or invoke a reporting tool to generate a report.

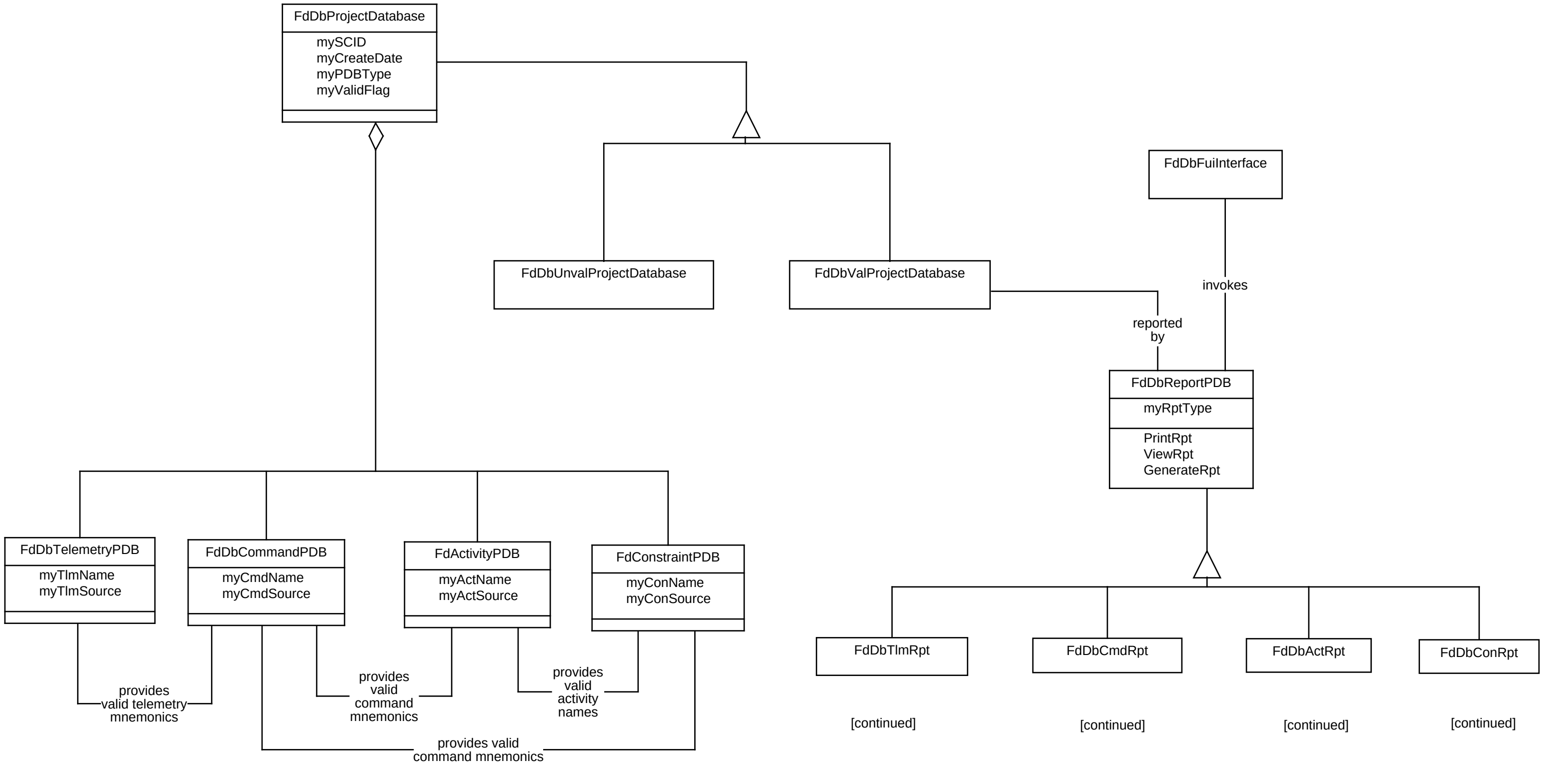


Figure 3.5-1. PDB Report Object Model

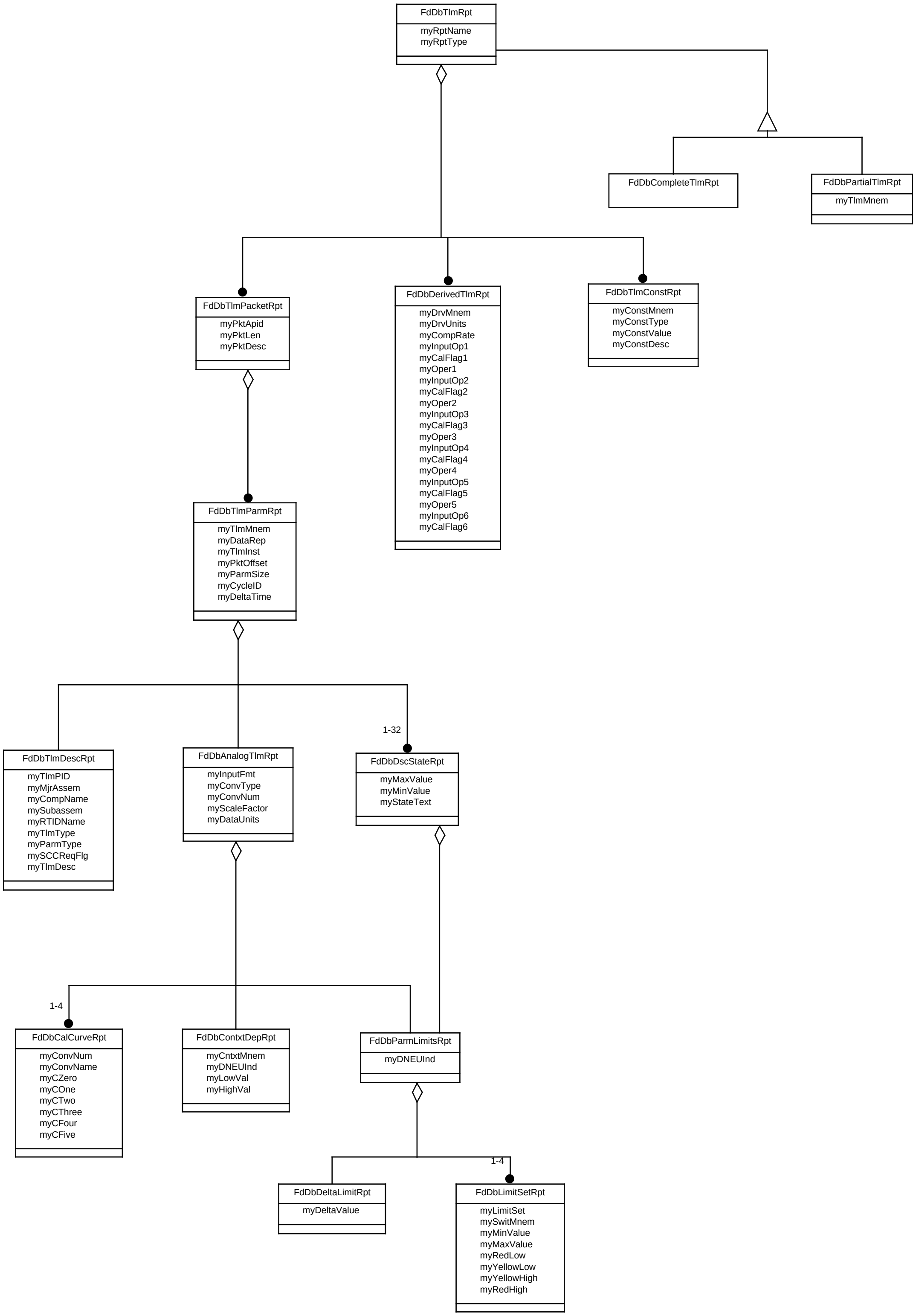


Figure 3.5-2. PDB Report Object Model

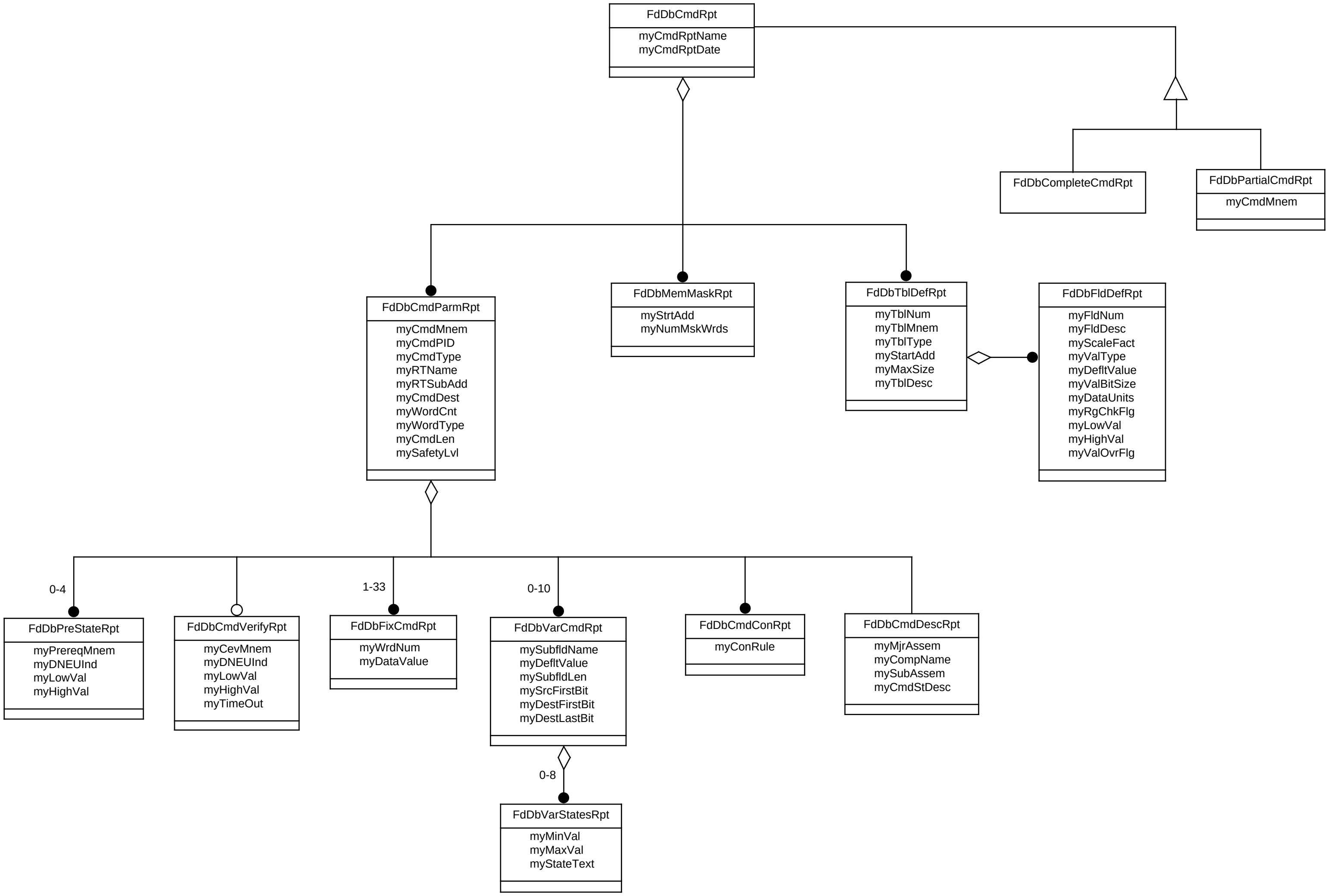


Figure 3.5-3. PDB Report Object Model

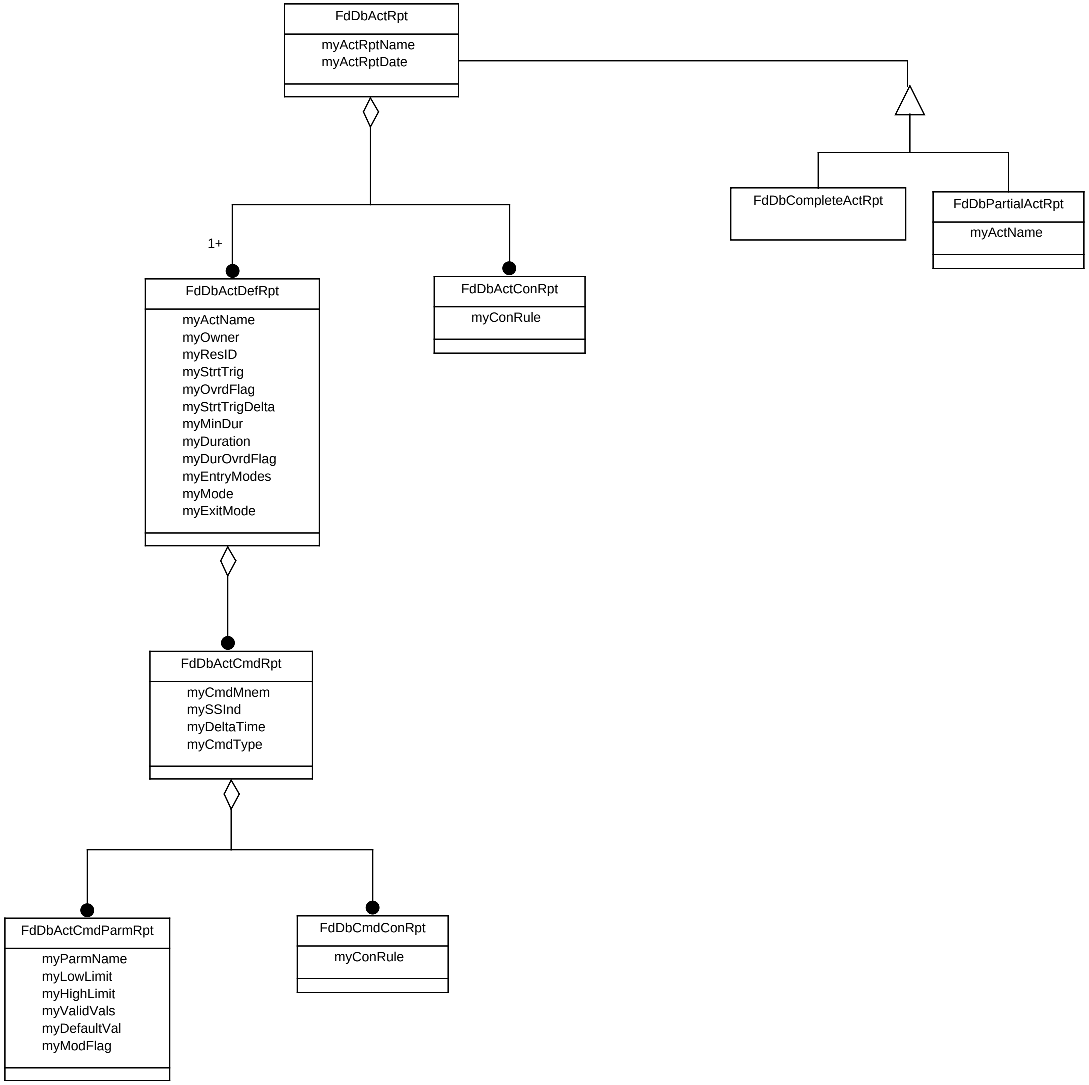


Figure 3.5-4. PDB Report Object Model

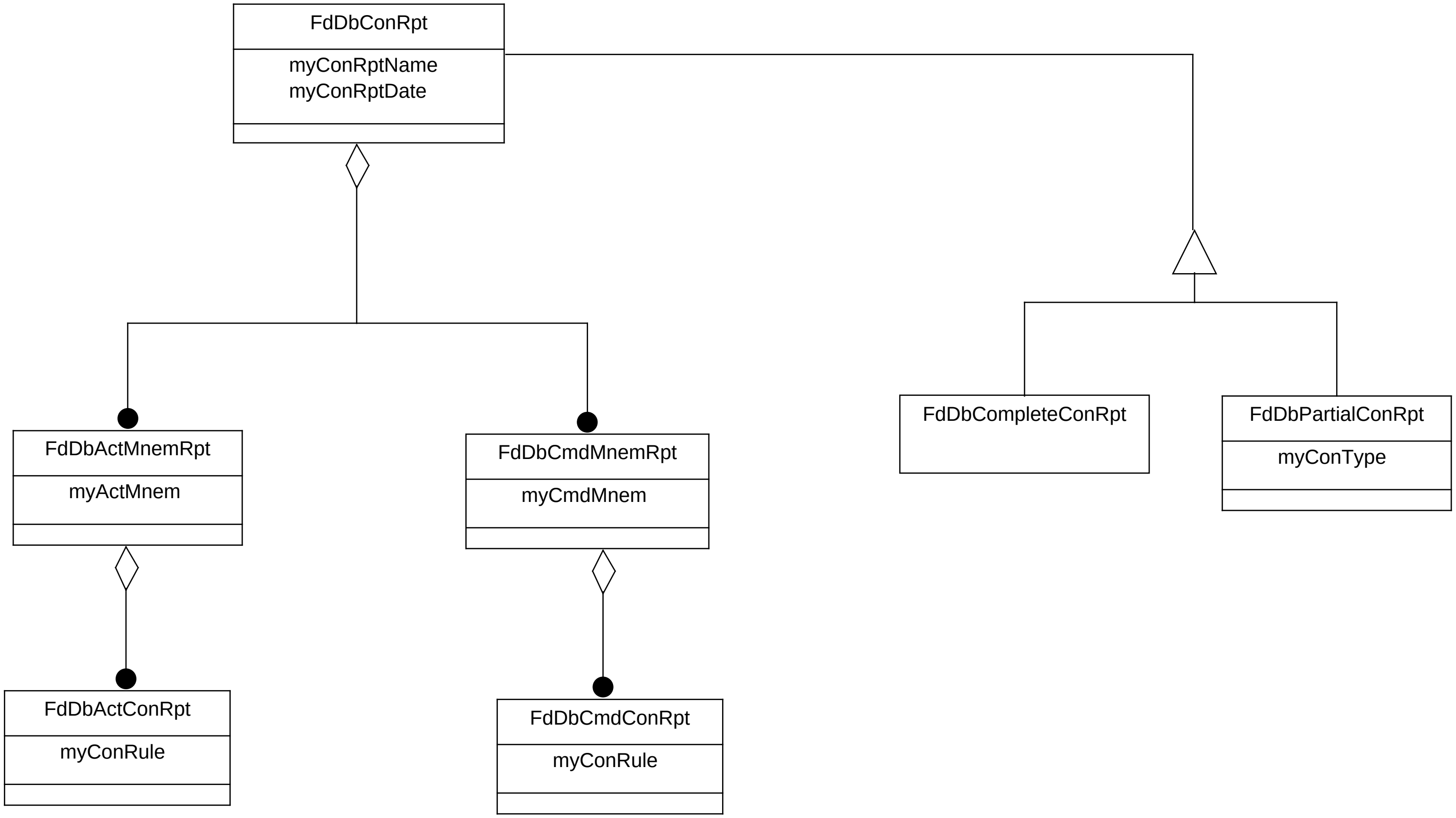


Figure 3.5-5. PDB Report Object Model

3.5.4 PDB Report Dynamic Model

3.5.4.1 PDB Report Scenario Abstract

3.5.4.2 PDB Report Summary Information

Interfaces:

User Interface

Stimulus:

DBA selection of the PDB Report option from the Database Utilities Menu

Desired Response:

Edits to the unvalidated PDB

Pre-Conditions:

The database is up and running.

The database is initialized.

The PDB definitions are ingested.

The PDB is validated.

Post-Conditions:

None

3.5.4.3 PDB Report Scenario Description

The Project Data Base (PDB) Report Generation process is initiated through the selection of the Project Data Base (PDB) Report Generation option on the Database Utilities Menu.

The user specifies which PDB data to report on based on User Interface prompts. Once the user has made data type selections, the database reporting tool is invoked. (The report is generated using a database COTS product.)

Once the report is generated, the database reporting tool is exited and a status message is displayed on the screen. The report may then be viewed or printed by selecting the appropriate options off of the User Interface screen.

The PDB Report Generation screen is exited by selecting an exit button on the screen or by selecting another option off of the Database Utiltiies Menu.

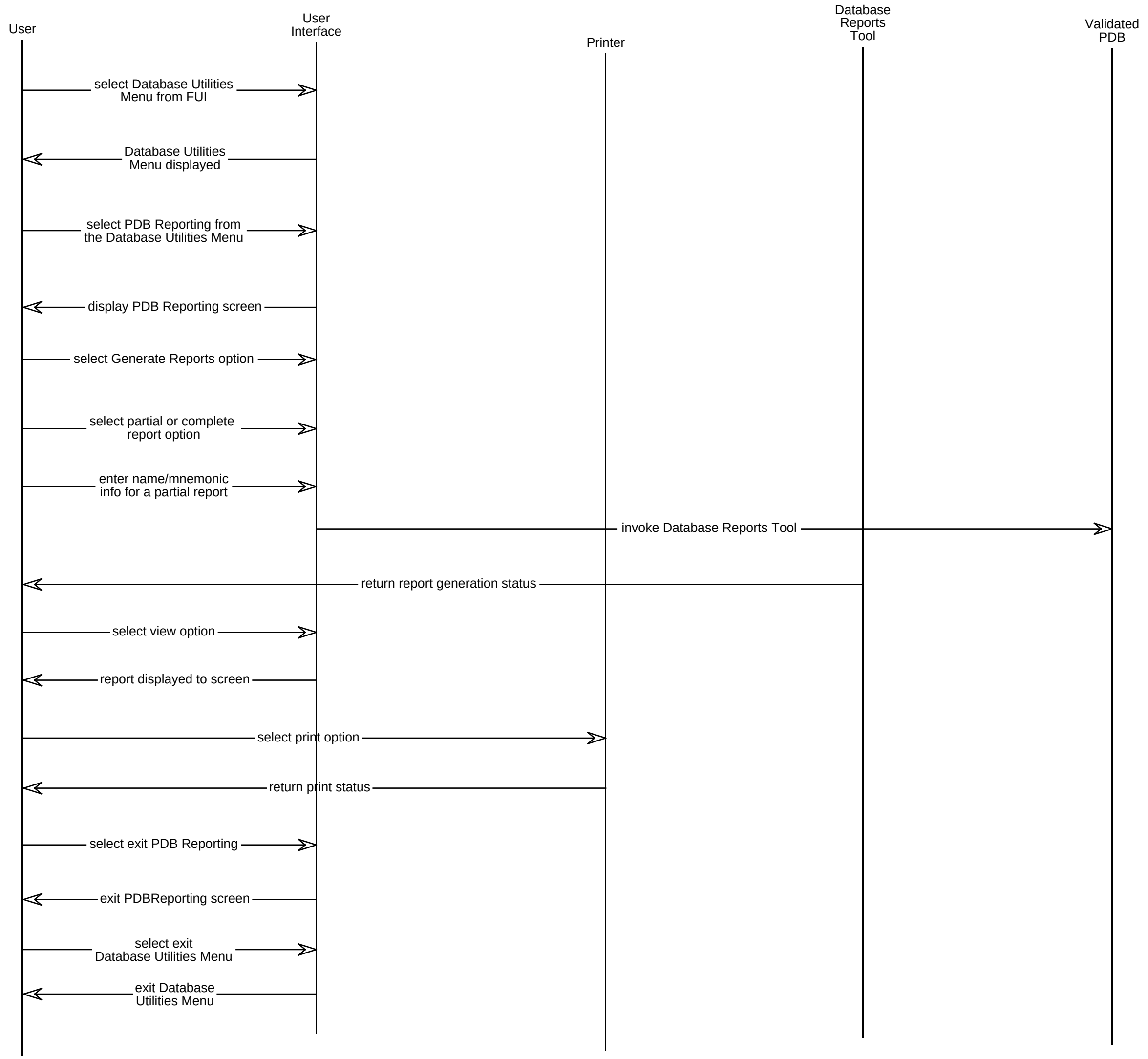

User
User Interface
Printer
Database Reports Tool
Validated PDB
select Database Utilities Menu from FUI
Database Utilities Menu displayed
select PDB Reporting from the Database Utilities Menu
display PDB Reporting screen
select Generate Reports option
select partial or complete report option
enter name/mnemonic info for a partial report
invoke Database Reports Tool
return report generation status
select view option
report displayed to screen
select print option
return print status
select exit PDB Reporting
exit PDBReporting screen
select exit Database Utilities Menu
exit Database Utilities Menu

Figure 3.5-6. PDB Report Event Trace

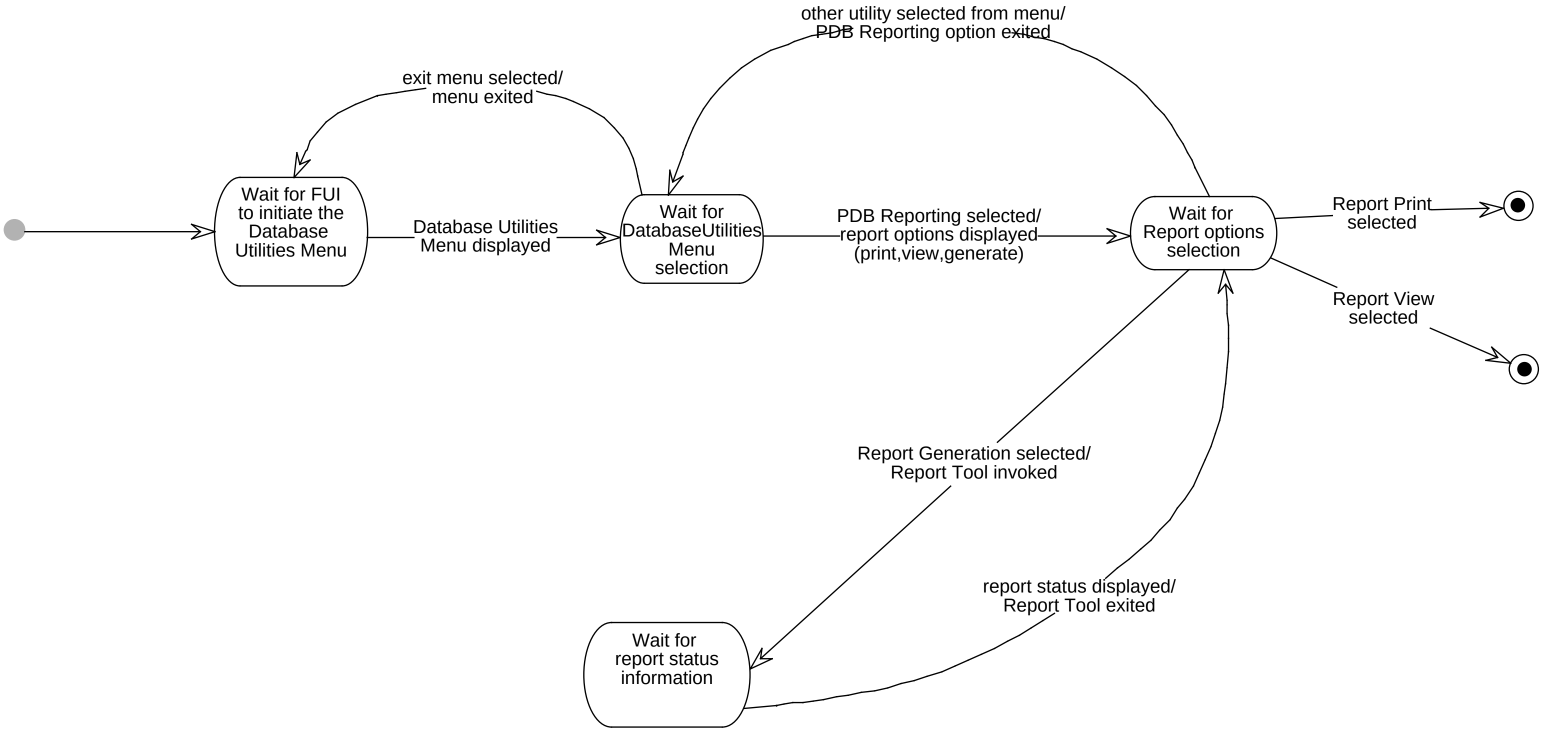

Wait for FUI to initiate the Database Utilities Menu
Database Utilities Menu displayed
Wait for DatabaseUtilities Menu selection
exit menu selected/ menu exited
other utility selected from menu/ PDB Reporting option exited
PDB Reporting selected/ report options displayed (print,view,generate)
Wait for Report options selection
Report Print selected
Report View selected
Report Generation selected/ Report Tool invoked
report status displayed/ Report Tool exited
Wait for report status information

Figure 3.5-7. PDB Report State Diagram

3.5.5 PDB Report Data Dictionary

Note: Refer to the DFCD for the EOS AM-1 PDB and the FOS Database Design and Database Schema Specifications for specific details supporting the design of PDB processing.

Class Name: FdDbActConRpt

Description: The Activity Constraint Report provides the activity constraints on an Activity Report.

Class Name: FdDbActCmdRpt

Description: The Activity Definition Report class provides the activity command definitionportion of the Activity report..

Class Name: FdDbActMnemRpt

Description: The Activity Mnemonic Report class represents the list of Activity Mnemonics on the Constraint Report.

Class Name: FdDbActRpt

Description: The Activity Report class represents the Activity Report.

Attributes:

myActRptName: string

Description: the name given to the report

myActRptDate: string

Description: the date the report was generated

Class Name: FdDbCmdConRpt

Description: The Command Constraint provides the command constraints on an Activity Report

Class Name: FdDbActCmdParmRpt

Description: The Activity Command Parameter Report provides activity Command Parameter portion of the Activity Report.

Class Name: FdDbActDefRpt

Description: The Activity Definition Report class provides the activity definition data of an Activity Report.

Class Name: FdDbCmdMnemRpt

Description: The Command Mnemonic Report class represents the list of Command Mnemonics on the Constraint Report.

Class Name: FdDbCmdRpt

Description: The Command Report class represents the Command Report.

Attributes:

myCmdRptName: string

Description: the name given to the report

myCmdRptDate: string

Description: the date the report was generated

Class Name: FdDbCompleteActRpt

Description: The Complete Activity Report class represents a complete ActivityReport, listing all information about all activities.

Class Name: FdDbCompleteCmdRpt

Description: The Complete Command Report class represents a complete Command Report, listing all information about all commands.

Class Name: FdDbCompleteConRpt

Description: The Complete Constraint Report class represents a complete ConstraintReport, listing all information about all constraints.

Class Name: FdDbCompleteTlmRpt

Description: The Complete Telemetry Report class represents a complete TelemetryReport, listing all information about all telemetry mnemonics.

Class Name: FdDbConRpt

Description: The Constraint Report class represents the Constraint Report.

Attributes:

myActRptName: string

Description: the name given to the report

myActRptDate: string

Description: the date the report was generated

Class Name: FdDbPartialActRpt

Description: The Partial Activity Report class represents a partial Activity Report, corresponding to the activity name specified.

Class Name: FdDbPartialCmdRpt

Description: The Partial Command Report class represents a partial Command Report, corresponding to the command mnemonic specified.

Class Name: FdDbPartialConRpt

Description: The Partial Constraint Report class represents a partial Constraint Report, corresponding to the constraint type specified.

Attributes:

myConType: string

Description: the type of constraint report, command or activity

Class Name: FdDbPartialTlmRpt

Description: The Partial Telemetry Report class represents a partial Telemetry Report, corresponding to the telemetry mnemonic specified.

Class Name: FdDbTlmRpt

Description: The Telemetry Report class represents the Telemetry Report.

Attributes:

myTlmRptName: string

Description: the name given to the report

myTlmRptDate: string

Description: the date the report was generated

Class Name: FdDbAnalogTlmRpt

Description: The Analog Telemetry Report class represents the analog telemetry information associated with a telemetry parameter on a Telemetry Report.

Class Name: FdDdCmdDescRpt

Description: The Command Description Report class represents the command description information associated with a command parameter for a Command Report.

Class Name: FDbCmdParmRpt

Description: The Command Parameter Report class represents the command parameter information on the Command Reprot.

Class Name: FdDbCmdVerifyRpt

Description: The Command Execution Verification (CEV) Report class represents the CEV information for a command parameter on a Command Report.

Class Name: FdDbDeltaLimitRpt

Description: The Delta Limit Report class represents the delta limit assoicated with a telemetry parameter limit on a Telemetry report.

Class Name: FdDbParmLimitsRpt

Description: The Parameter Limits Report class represents the limits assoicated with a telemetry parameter on a Telemetry report.

Class Name: FdDbDscStateRpt

Description: The Discrete States Report class represents the discrete state information for a telemetry parameter on a Telemetry Report.

Class Name: FdDbFixCmdRpt

Description: The Fixed Data Word Report class represents fixed command information associated with a command parameter on a Command Report.

Class Name: FDbFldDefRpt

Description: The Table Field Definition Record represent the field defintion data for the table definition data on the Command Report.

Class Name: FdDbLimitSetRpt

Description: The Limit Set Report class represents the limits set information associated with a telemetry parameter limit on a Telemetry Report.

Class Name: FdDbMemMaskRpt

Description: The Memory Masking Definition Report represents the memory mask information on the Command Report.

Class Name: FdDbPreStateRpt

Description: The Prerequisite State Report class represents the prerequisite state information for a command parameter on a Command Report.

Class Name: FdDbReportPDB

Description: The Report PDB class represents the PDB Report.

Attributes:

myRptType: string

Description:report type indicated the type of report (telemetry, command, activity, or constraint)

Operations:

FdDbReportPDB :: PrintRpt

Description:the operation to print a generated report

FdDbReportPDB :: ViewRpt

Description:the operation to view a generated report

FdDbReportPDB :: GenerateRpt

Description:the operation to generate a report

Class Name: FdDbTblDefRpt

Description: The Table Definition Record represents the table definition information on the Command Report.

Class Name: FdDbTlmDescRpt

Description: The Telemetry Description Report class represents the telemetry description information assoicated with a telemetry parameter on a Telemetry Report.

Class Name: FdDbTlmPacketRpt

Description: The Telemetry Packet Report represents the telemetry packet information on a Telemetry Report.

Class Name: FdDbTlmParmRpt

Description: The Telemetry Parameter Report class represents the telemetry parameter information on a Telemetry Report.

Class Name: FDbVarCmdRpt

Description: The CommandVariable Command Report class represents the variable command information associated with a command paramater on a Command Report.

Class Name: FdDbVarStatesRpt

Description: The Variable States Report class represents the variable states for a variable command assoicated with a command parameter for a Command Report.

3.6 Operational Data Generation

3.6.1 Operational Data Generation Context

Refer to Section 3.2.1

3.6.2 Operational Data Generation Interfaces

Refer to Section 3.2.2

3.6.3 Operational Data Generation Object Model

FdDbOperationalData represents the data generated and maintained by the DMS. This information is used to support FOS operations. It is made up of telemetry, command, constraint and activity data (FdDbTelemetryOpData, FdDbCommandOpData, FdDbConstraintOpData, FdDbActivity-OpData). Operational data may be maintained as database tables or in UNIX files.

Each of the operational data generation subclasses (FdDbTlmOpDataGen, FdDbCmdOpDataGen, FdDbConOpDataGen, FdDbActOpDataGen) is derived from the FdDbGenOpData base class. They are responsible for controlling the conversion of each type of PDB data into an operational format. Upon acceptance of the validated PDB, the DBA will invoke this process to produce a new version of the operational data.

The class FdDbTelemeteryOpData represents the information that is used to support telemetry processing during FOS operations. The FdDbFUITlmODF controls the generation of operational data files used by the FOS User Interface Subsystem. The class FdDbTlmSubsysODF provides a listing of telemetry subsystem names and is made up of the subclass FdDbTlmSubsysDef. The class FdDbTlmMnemODF provides a listing of valid telemetry mnemonics associated with the current version of operational telemetry data. It is made up of the subclass FdDbTlmMnemDef. FdDbSCTlmODF controls the generation of the operational data files used to support telemetry processing. The class FdDbTlmParmODF is made up of the subclasses that represent the telemetry parameter definitions (FdDbTlmPktDef, FdDbTlmParmDef, FdDbAnaTlmDef, FdDbDiscTlmDef, FdDbConversion, FdDbTlmLimits, FdDbDeltas, FdDbBndryGrp, FdDbStates).

The class FdDbCommandOpData represents the information that is used to support spacecraft commanding during FOS operations. FdDbCEVODF controls the generation of the operational command execution verification file for the Command Management Subsystem. It is made up of the subclass FdDbCEVDEF. An instance of this subclass contains the execution verification criteria for a command mnemonic. FdDbCmdODF controls the generation of the operational command parameter file used to support Real-time commanding. It is made up of the subclasses FdDbCommandParm, FdDbPreState, FdDbFixData, FdDbVarData, FdDbVarConv, FdDbVarStates. The class FdDbOpCmdDB controls the creation of the operational database tables for commanding. The Planning & Scheduling Subsystem and Command Management Subsystem interface directly with the FOS Database during operations and must have access to this information. The subclass FdDbCommandODT represents the command operational data tables which is made up of the class FdDbCommandPDB.

The class FdDbOpActDB controls the creations of the operational database tables for activity definitions. The Planning & Scheduling Subsystem and the Command Management Subsystem interface directly with the FOS Database during operations and must have access to this information.

The subclass FdDbActivityODT represents the activity operational data tables which is made up of the class FdDbActPDB.

The class FdDbOpConDB controls the creation of the operational database tables for constraint definitions. The Planning & Scheduling Subsystem and the Command Management Subsystem interface directly with the FOS Database during operations and must have access to this information. The subclass FdDbConstraintODT represents the constraint operational data tables which is made u p of the class FdDbConstraintPDB.

3.6.4. Operational Data Generation Dynamic Model

3.6.4.1 Operational Data Generation Scenario Abstract

The Operational Data Generation scenario describes the generation of operational data tables and files used to support FOS operations.

3.6.4.2 Operational Data Generation Summary Information

Interfaces:

None

Stimulus:

DBA selection of the operational data generation option from the Database Utilities menu

Desired Response:

The creation of the operational data tables and files from telemetry, command, constraint and activity PDB.

Pre-Conditions:

FOS Database initialized.

PDB validated.

Post-Conditions:

None

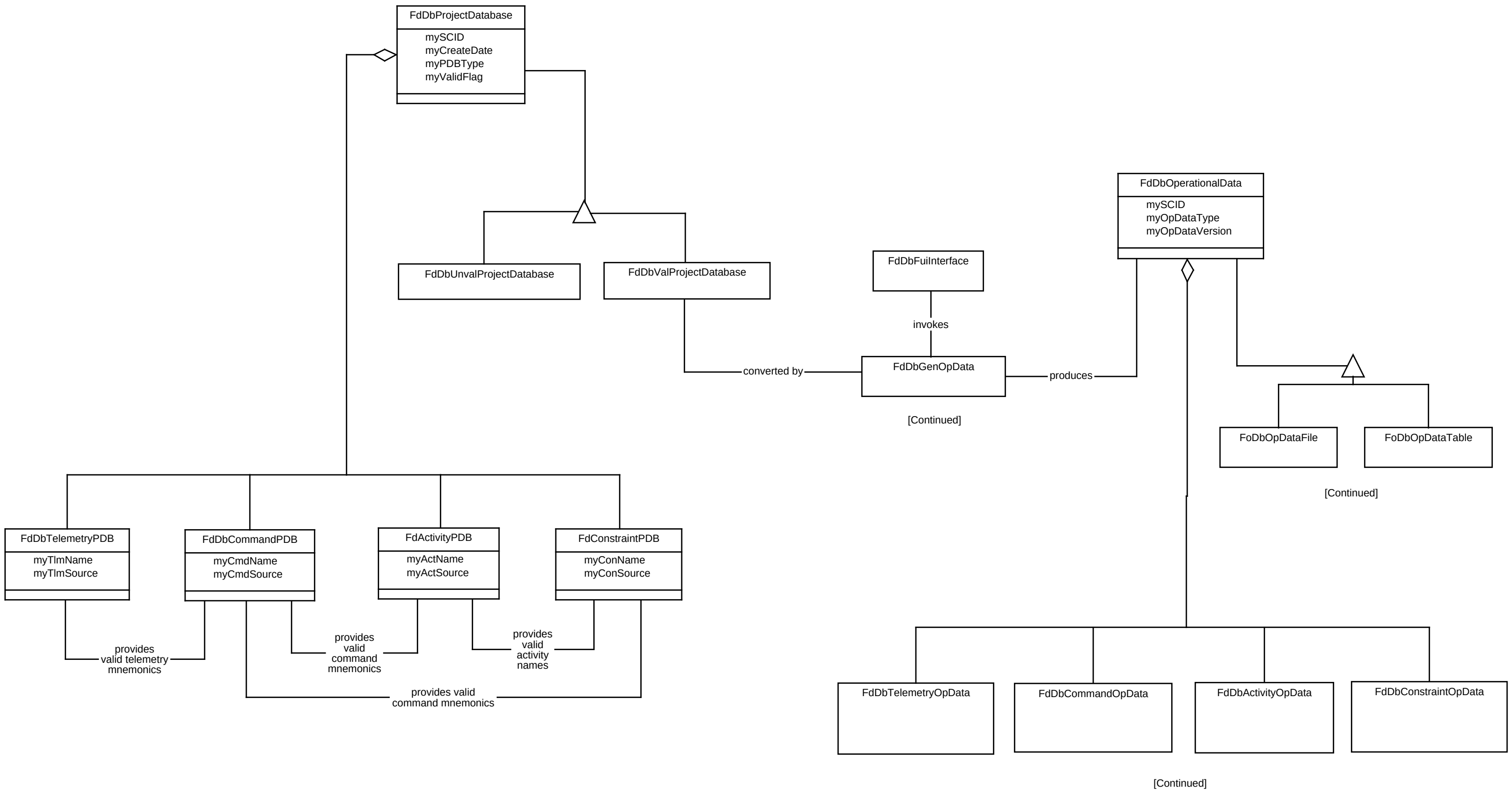


Figure 3.6-1. Operational Data Generation Object Model

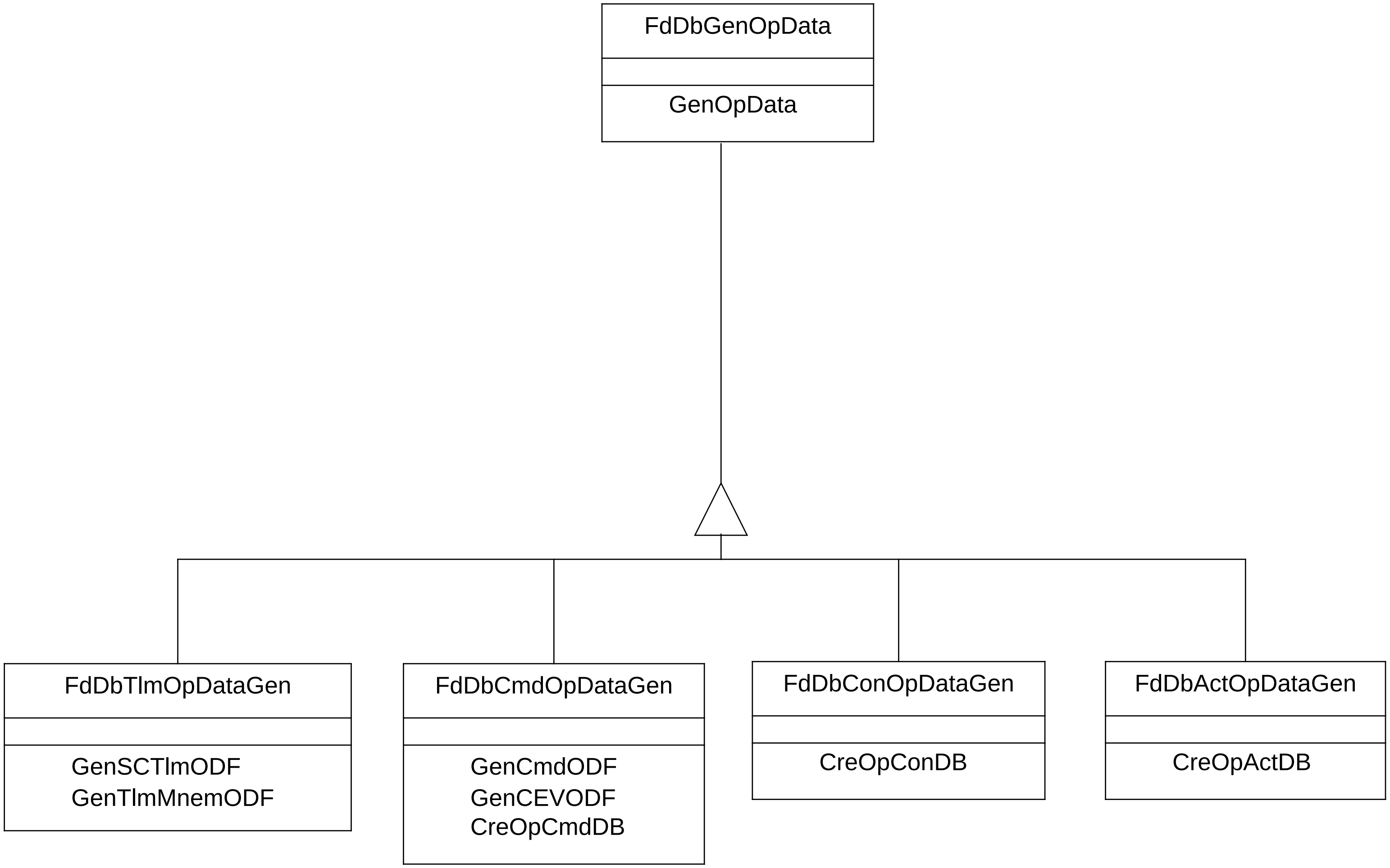


Figure 3.6-2. Operational Data Generation Object Model

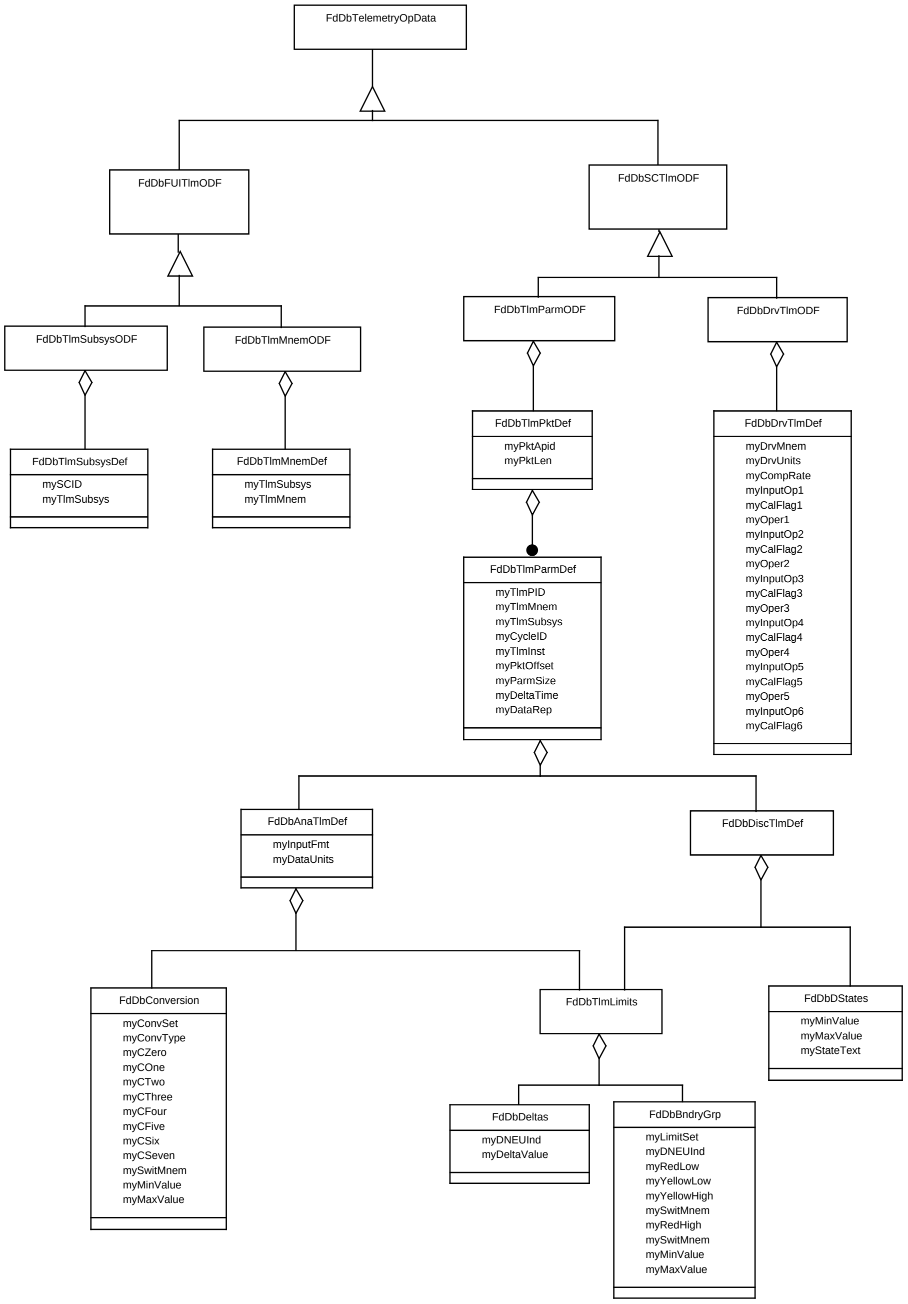


Figure3.6-3. Operational Data Generation Object Model

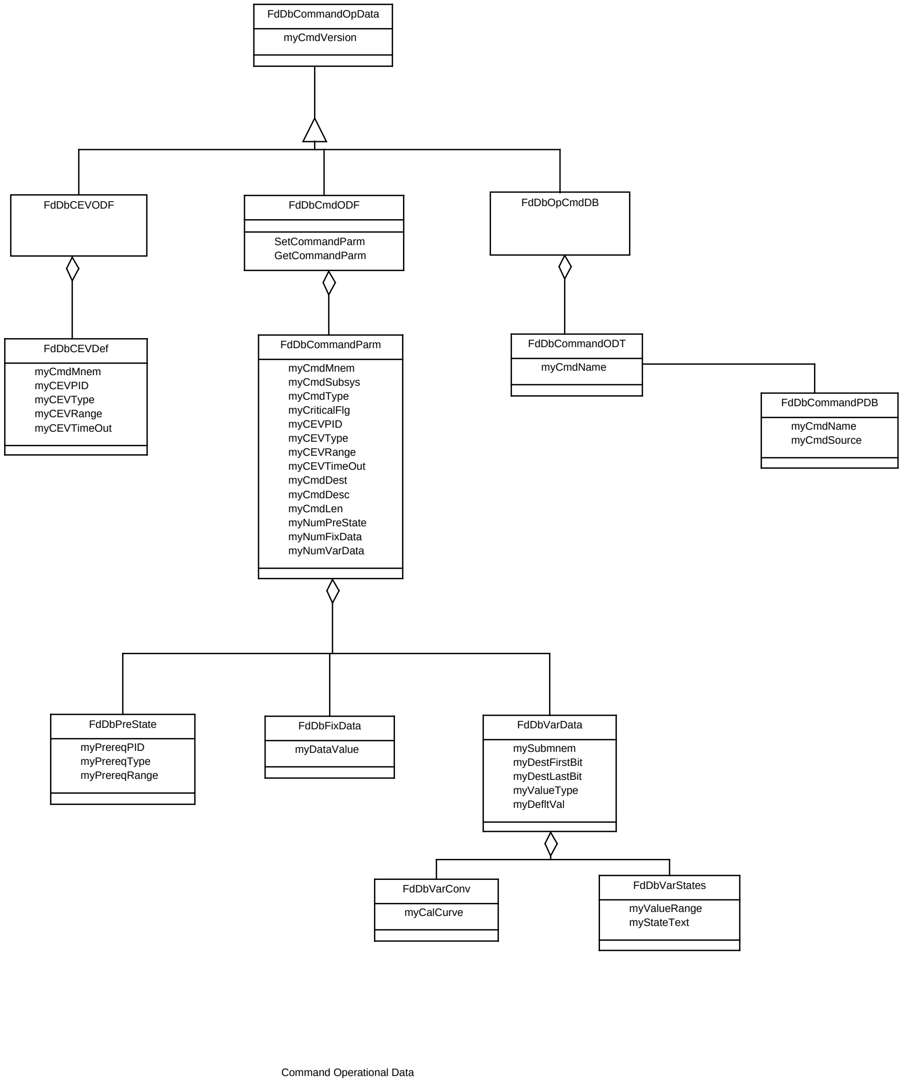


Figure 3.6-4. Operational Data Generation Object Model

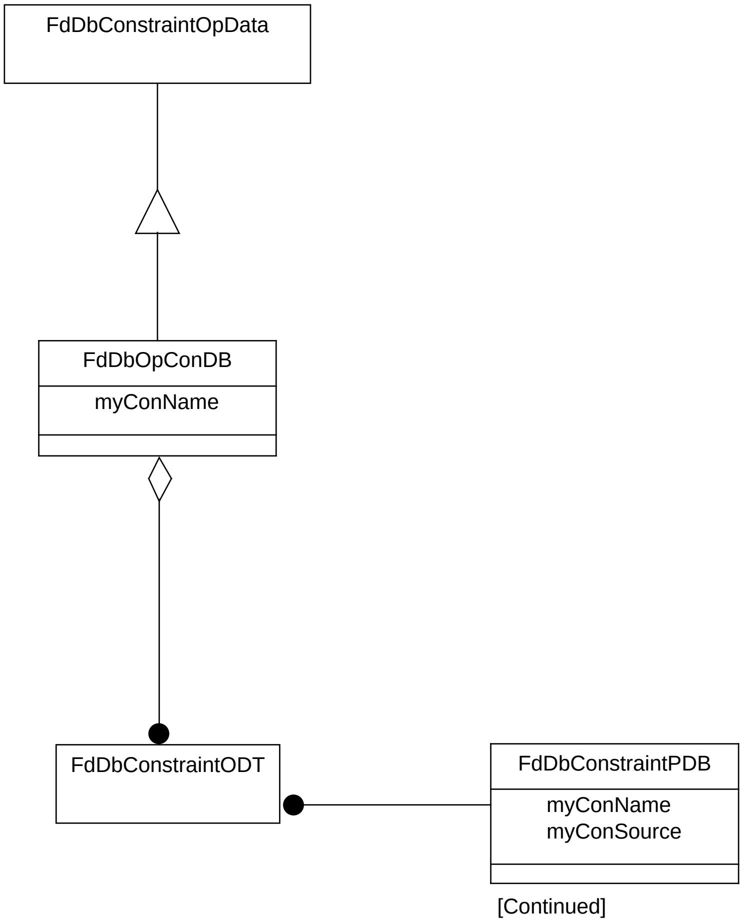
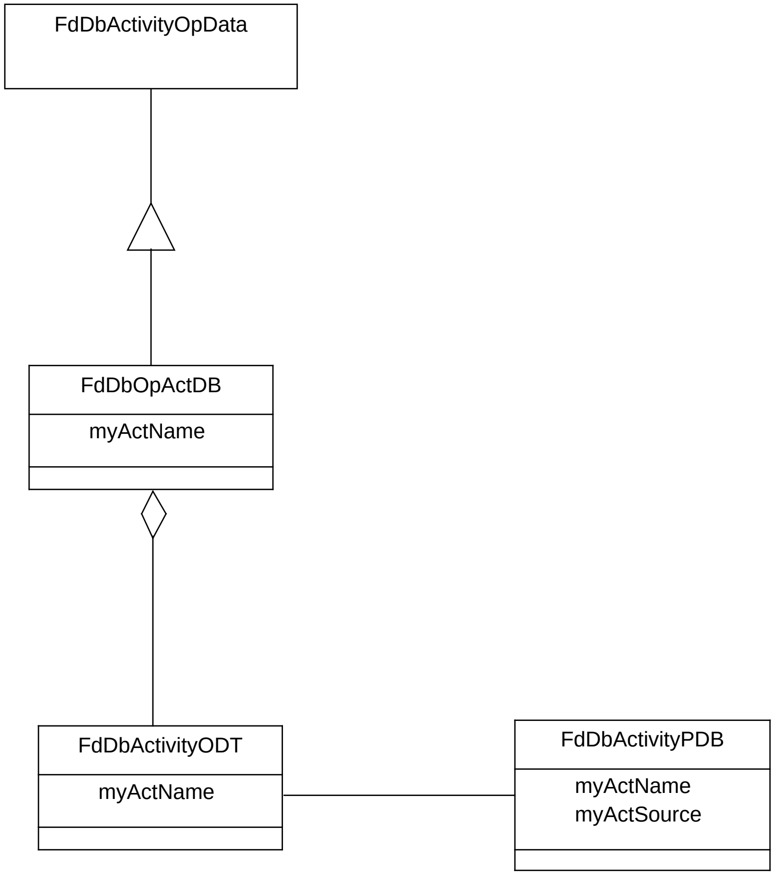


Figure 3.6-5. Operational Data Generation Object Model



[Continue]

Figure 3.6-6. Operational Data Generation Object Model

3.6.4.3 Operational Data Generation Scenario Description

The Operational Data Generation process is initiated through the selection of the operational data generation option on the Database Utilities menu by the DBA.

The generation of the telemetry operational data is invoked first. Telemetry definitions are copied from the validated PDB within the database and put into a format useful for telemetry processing. This information is stored in UNIX files, which include the Telemetry Subsystem ODF, the Telemetry Mnemonic ODF, the Telemetry Parameter ODF and the Derived Telemetry ODF.

Next, the creation of the operational command data is invoked. The definitions from the validated command PDB are copied into the command operational data tables. Additionally, the Command Execution Verification ODF and the Command Parameter ODF are created by copying the command definitions from the database tables into a UNIX file.

The constraint and activity operational data generation includes copying the validated constraint and activity PDB into an operational area for access by Planning & Scheduling the Command Management Subsystems.

Upon completion of the generation process a new version of the operational data is made available by the Data Management Subsystem for use by the FOS Subsystems.

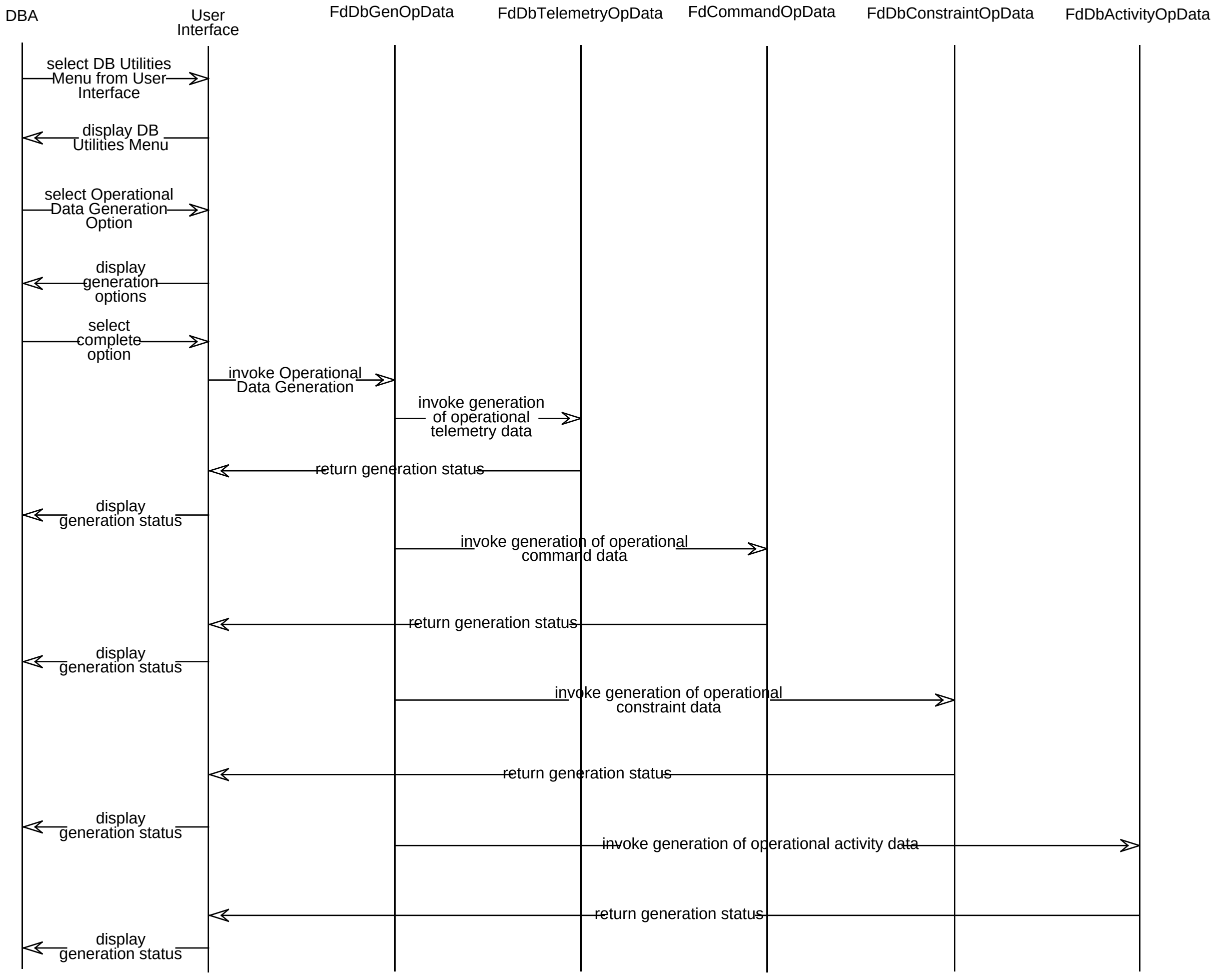

DBA
User Interface
FdDbGenOpData
FdDbTelemetryOpData
FdCommandOpData
FdDbConstraintOpData
FdDbActivityOpData
select DB Utilities Menu from User Interface
display DB Utilities Menu
select Operational Data Generation Option
display generation options
select complete option
invoke Operational Data Generation
invoke generation of operational telemetry data
return generation status
display generation status
invoke generation of operational command data
return generation status
display generation status
invoke generation of operational constraint data
return generation status
display generation status
invoke generation of operational activity data
return generation status
display generation status

Figure 3.6-7. Operational Data Generation Event Trace

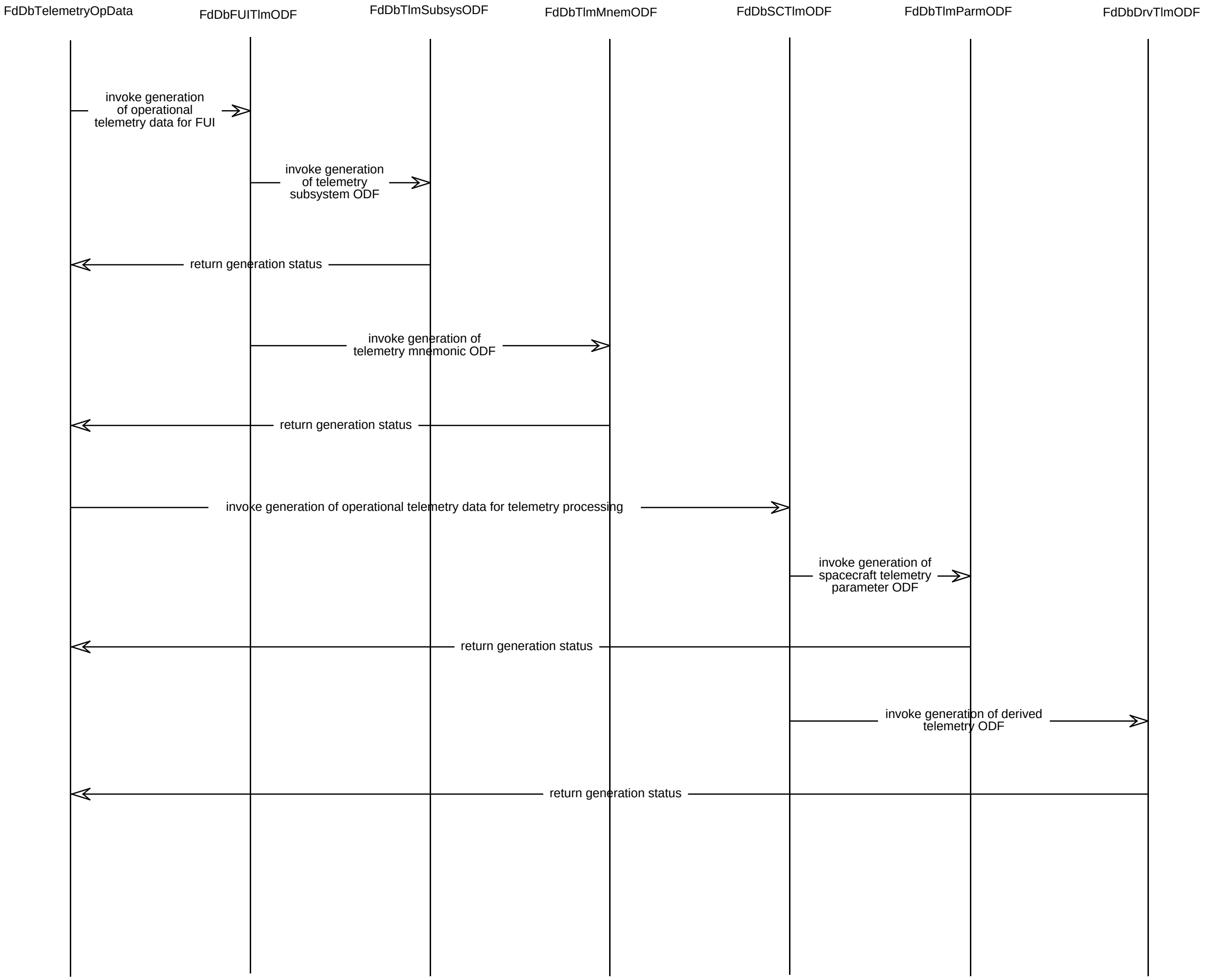

FdDbTelemetryOpData
FdDbFUITlmODF
FdDbTlmSubsysODF
FdDbTlmMnemODF
FdDbSCTlmODF
FdDbTlmParmODF
FdDbDrvTlmODF
invoke generation of operational telemetry data for FUI
invoke generation of telemetry subsystem ODF
return generation status
invoke generation of telemetry mnemonic ODF
return generation status
invoke generation of operational telemetry data for telemetry processing
invoke generation of spacecraft telemetry parameter ODF
return generation status
invoke generation of derived telemetry ODF
return generation status

Figure 3.6-8. Operational Data Generation Event Trace

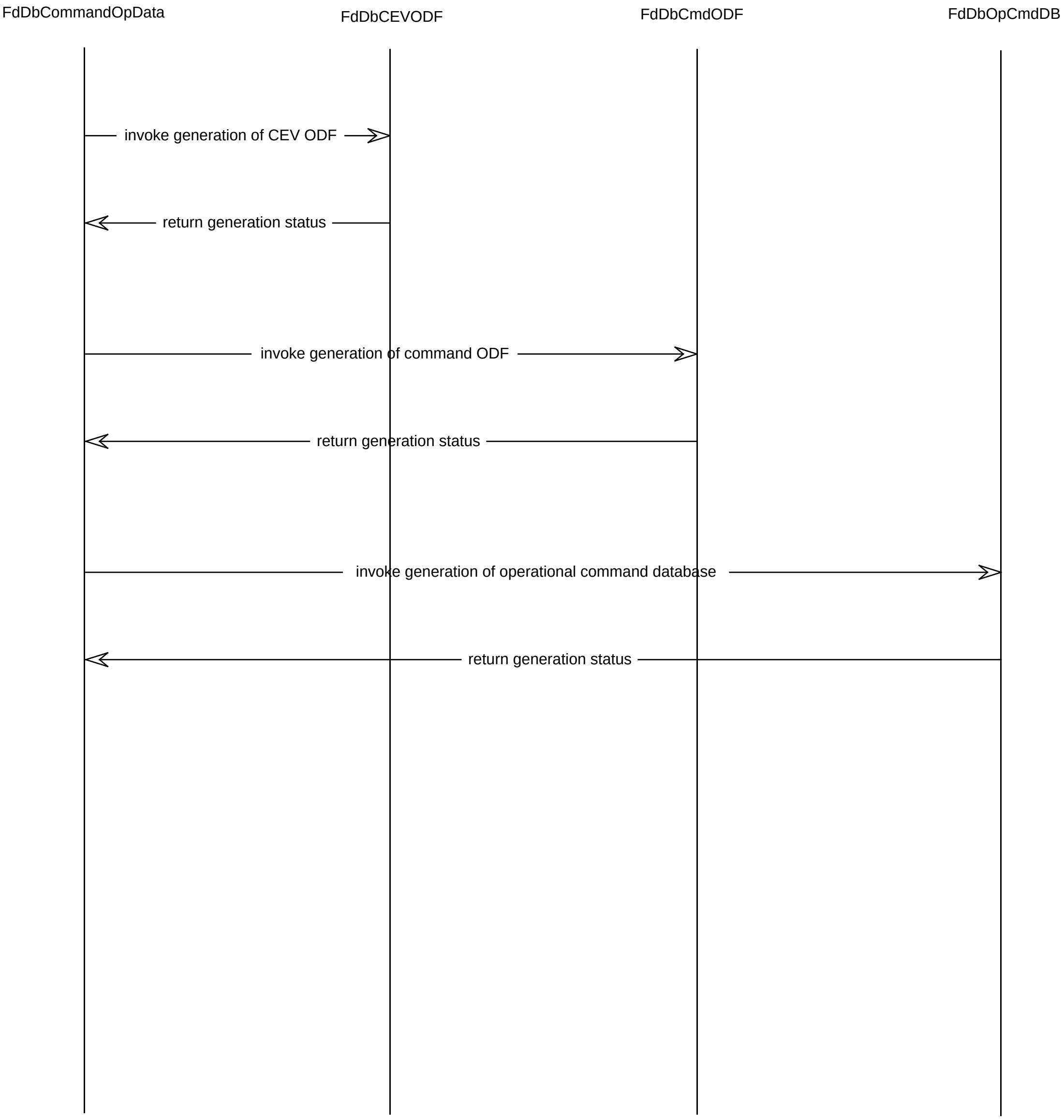


Figure 3.6-9. Operational Data Generation Event Trace

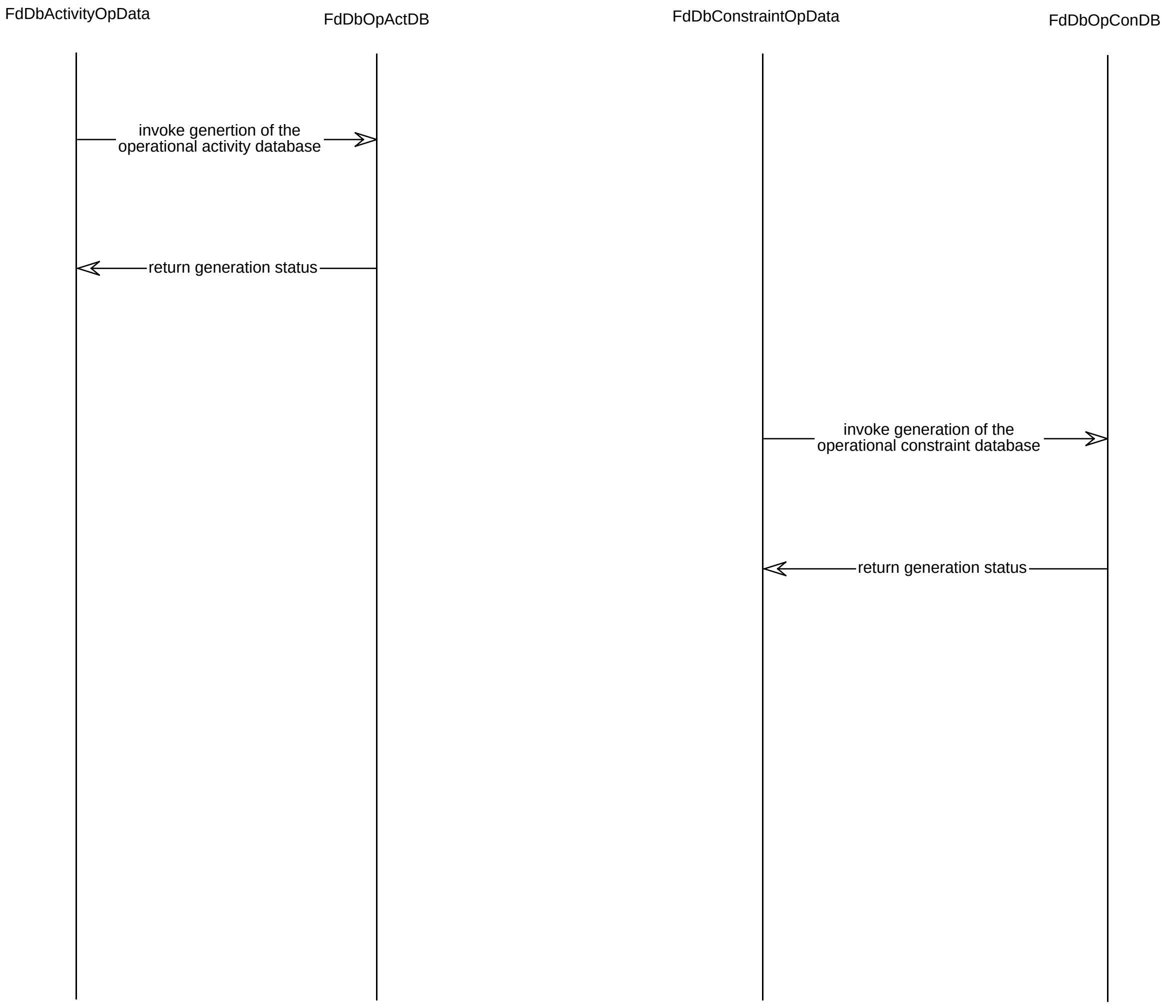

FdDbActivityOpData
FdDbOpActDB
FdDbConstraintOpData
FdDbOpConDB
invoke genertion of the operational activity database
return generation status
invoke generation of the operational constraint database
return generation status

Figure 3.6-10. Operational Data Generation Event Trace

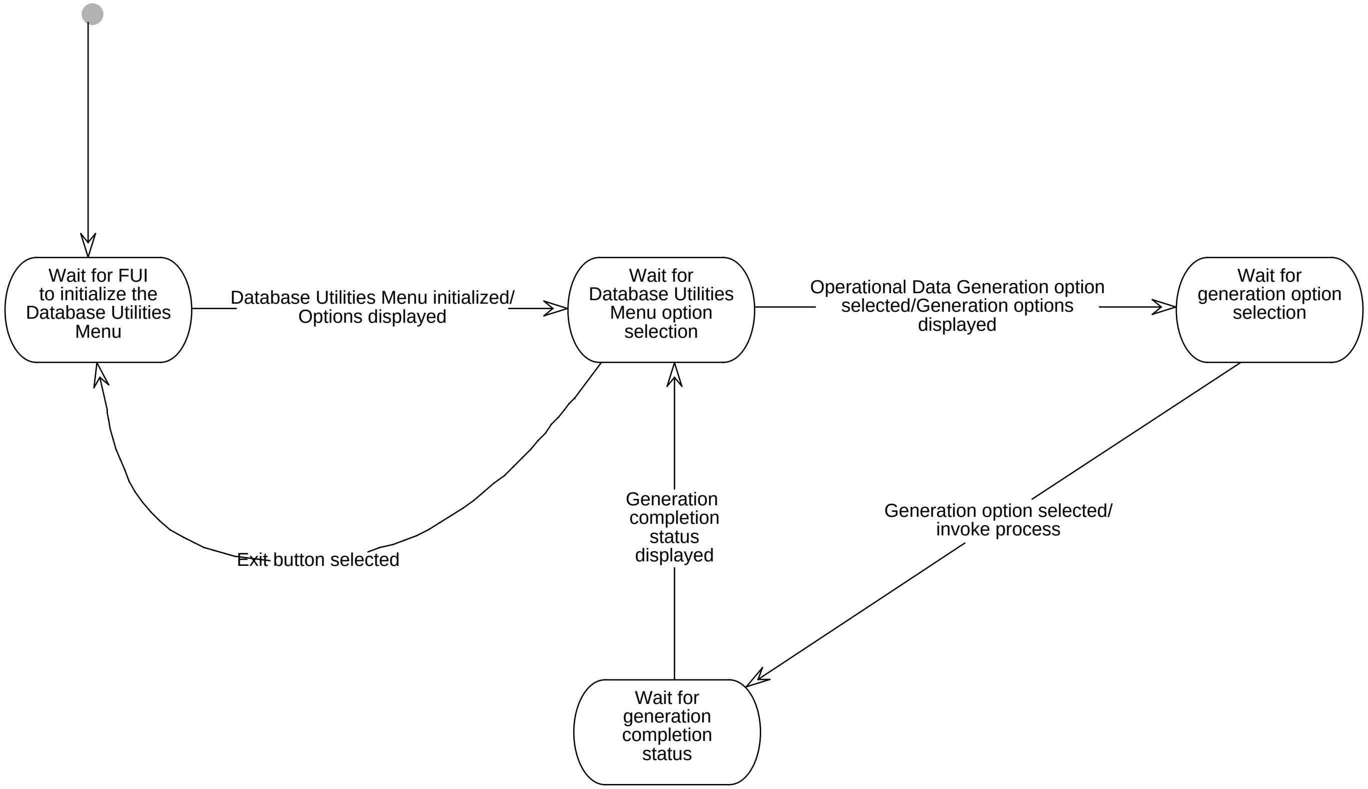

Wait for FUI to initialize the Database Utilities Menu
Database Utilities Menu initialized/ Options displayed
Wait for Database Utilities Menu option selection
Operational Data Generation option selected/Generation options displayed
Wait for generation option selection
Exit button selected
Generation completion status displayed
Generation option selected/ invoke process
Wait for generation completion status

Figure 3.6-11. Operational Data Generation State Diagram

3.6.5 Operational Data Generation Data Dictionary

Note: Refer to the DFCD for the EOS AM-1 PDB and the FOS Database Design and Database Schema Specifications for specific details supporting the design of PDB processing.

Class Name: FdDbActOpDataGen

The Activity Operational Data Generation class provides the functional operations needed to generate the operational constraint data.

Operations:

FdDbActOpDataGen:: CreOpActDb

operation to create the operational activity database.

Class Name: FdDbActivityODT

The Activity ODT class represents operational activity database tables used in support of FOS operations for mission planning and spacecraft commanding.

Class Name: FdDbActivityOpData

The Activity Operational Data class contains activity information used during FOS operations to support the Planning & Scheduling and Command Management Subsystems..

Class Name: FdDbActivityPDB

The Activity PDB class represents the activity definition files needed to support FOS operations.

Class Name: FdDbAnaTlmDef

The Analog Telemetry Definition class provides characteristic information about analog telemetry parameters.

Class Name: FdDbBndryGrp

The Boundary Group class contains the red/yellow - high/low limit checking criteria associated with an analog or discrete telemetry parameter.

Class Name: FdDbCEVDef

The CEV Definition class provides the criteria used to verify execution of a command during FOS operations.

Class Name: FdDbCEVODF

The CEV ODF class contains command executions verification definitions in support of the Command Management Subsystem during FOS operations.

Class Name: FdDbCmdODF

The Command ODF class contains the command definitions used to support spacecraft commanding during FOS operations.

Class Name: FdDbCmdOpDataGen

The Command Operational Data Generation class provides operations used to generate the operational command data.

Operations:

FdDbCmdOpDataGen::GenCmdODF

Description:operation to generate the command operational data file used in support of real-time Commanding.

FdDbCmdOpDataGen::GenCEVODF

Description:operation to generate the command execution verification list for the Command Management Subsystem.

FdDbCmdOpDataGen::CreOpCmdDB

Descriptionoperation to create the operational command database.

Class Name: FdDbCommandODT

The Command ODT class represents operational command database tables used in support of FOS operations for the Planning & Scheduling and Command Management Subsystems.

Class Name: FdDbCommandOpData

The Command Operational Data class contains command definitions in an operational format used support FOS operations.

Class Name: FDbCommandParm

The Command Parameter class contains an instance of a spacecraft or instrument command which is used to support real-time commanding of the EOS AM-1 spacecraft.

Class Name: FdDbCommandPDB

The Command PDB class represents the command definition files needed to support commanding of the EOS AM-1 spacecraft.

Class Name: FdDbConOpDataGen

The Constraint Operational Data Generation class provides the functional operations needed to generate the operational constraint data.

Operations:

FdDbConOpDataGen:: CreOpConDb

Description:operation to create the operational constraint database.

Class Name: FdDbConstraintODT

The Constraint ODT class represents operational constraint database tables used in support of FOS operations.

Class Name: FdDbConstraintOpData

The Constraint Operational Data class contains operational constraint data used in support of FOS operations.

Class Name: FdDbConstraintPDB

The Constraint PDB represent the constraint definition files needed to support constraint checking for commands and activities during FOS operations.

Class Name: FdDbConversion

The Conversion class provides the coefficients used to convert raw telemetry values into EUs.

Class Name: FdDbDeltas

The Deltas class provides the delta limit definition for a telemetry parameter

Class Name: FdDbDiscTlmDef

The Discrete Telemetry Definition class provides characteristic information about discrete telemetry parameters.

Class Name: FdDbDrvTlmDef

The Derived Telemetry Definition class contains a simple equations that combine previously defined analogs, discretes, constants and other derived parameters via arithmetic or logical functions.

Class Name: FdDbDrvTlmODF

The Derived Telemetry ODF class contains the derived telemetry parameter definitions.

Class Name: FdDbDState

The Discrete States class contains the association of a single text state to a range of values for a discrete telemetry parameter.

Class Name: FDbFixData

The Fixed Data Word class represents a fixed data word associated with a command.

Class Name: FdDbFUITlmODF

The FUI Telemetry ODF class represents the telemetry data files generated to support FOS User Interface.

Class Name: FdDbGenOpData

The Generate Operational Data class provides an operation responsible for invoking the process for generating operational data.

Operations:

FdDbGenOpData::GenOpData

Description:operation to generate operational data for mission planning, space craft commanding and telemetry processing.

Class Name: FdDbOpActDB

The Operational Activity Database class represents the operational activity information maintained in table format in a COTS DBMS.

Class Name: FdDbOpCmdDB

The Operational Command Database class represents the operational command information maintained in table format in a COTS DBMS.

Class Name: FdDbOpConDB

The Operational Constraint Database class represents the operational constraint information maintained in table format in a COTS DBMS.

Class Name: FdDbOpDataFile

The Operational Data File class represents the operational information maintained by the DMS in a UNIX file format.

Class Name: FdDbOpDataGen

The Operational Data Generation class represents the operational information maintained by the DMS in a UNIX file format.

Class Name: FdDbOpDataTable

The Operational Data Table class represents the operational information maintained by the DMS in a COTS database product in table format.

Class Name: FdDbOperationalData

The Operational Data class represents the database tables and UNIX files that are used to support mission planning, spacecraft commanding and telemetry processing for the FOS.

Class Name: FdDbProjectDatabase

The Project Database class represents the telemetry, command, constraint and activity definitions files needed to support FOS operations.

Class Name: FdDbPreState

The Prerequisite State Specification Record defines the condition for which a telemetry parameter associated with a command must occur in order to perform prerequisite state checking.

Class Name: FdDbSCTlmODF

The Spacecraft Telemetry ODF class provides the validated telemetry PDB in an operational format to be used to support telemetry processing.

Class Name: FdDbTelemetryOpData

The Telemetry Operational Data class contains operational telemetry data used in support of FOS operations.

Class Name: FdDbTelemetryPDB

The Telemetry PDB class represents the telemetry definition files needed to support telemetry processing during FOS operations.

Class Name: FdDbTllmLimits

The Telemetry Limits class represents the telemetry limits definitions.

Class Name: FdDbTlmMnemDef

The Telemetry Mnemonic Definition class provides the current telemetry mnemonics used during operations.

Class Name: FdDbTlmMnemODF

The Telemetry Mnemonic ODF class contains the telemetry mnemonics supporting telemetry operations and is used by the FOS User Interface Subsystem.

Class Name: FdDbTlmOpDataGen

The Telemetry Operational Data Generation class provides operations used to generate the operational telemetry data.

Operations:

FdDbTlmOpDataGen::GenSCTlmODF

Description:operation to generate the spacecraft telemetry files for telemetryprocessing.

FdDbTlmOpDataGen::GenTlmMnemODF

Description:operation to generate the telemetry subsystem and mnemonic files to support FOS User Interface.

Class Name: FdDbTlmPktDef

The Telemetry Packet Definition class provides telemetry packet definitions in support of telemetry operations.

Class Name: FdDbTlmParmDef

The Telemetry Parameter Definition class provides the operational telemetry parameter definition used to support telemetry decommutation.

Class Name: FdDbTlmParmODF

The Telemetry Parameter ODF class provides the operational telemetry data used to support telemetry decommutation.

Class Name: FdDbTlmSubsysODF

The Telemetry Subsystem ODF class contains telemetry subsystem information for use by the FOS User Interface Subsystem.

Class Name: FdDbTlmSubsysDef

The Telemetry Subsystem Definition class provides the name of each telemetry subsystem supported by the spacecraft.

Class Name: FdDbUnvalProjectDatabase

The Unvalidated Project Database class represents the PDB in a state prior to having validation performed on it's contents.

Class Name: FdDbValProjectDatabase

The Validated Project Database class represents the PDB in a state after having validation performed on it's contents.

Class Name: FDbVarConv

The Command Variable Conversion class contains the conversion equation associated with variable type command.

Class Name: FDbVarData

The Command Variable Data class contains the subfields associated with variable type commands.

Class Name: FdDbVarStates

The Variable States class provides the states associated with a subfield.

3.7 DMS Event Processing

The event handler receives all network, system, and operational events. The event handler is responsible for sending unformatted events to the event archiver at the data server. The event archiver is responsible for formatting events using the unformatted event and the events database. The formatted event will contain UTC time of event, event type, event identifier, event message, instrument identifier, spacecraft identifier, event message, severity, filename where event occurred, and line number of the file. The event archiver will archive the event, multicast the event so that the event can be viewed by user stations, and initiate procedures (triggers).

The user has the option of configuring incoming and outgoing event filters. Both the event listener and the event handler will read an event filter configuration file during initialization. Outgoing event filters prevents the event handler from sending duplicated events over the network (i.e., tlm limits), and incoming event filters directs the event listener to only listen for selected event types.

3.7.1 DMS Event Processing Context

The DMS event processing interfaces with all FOS subsystems and with the MSS, as shown in the Context Diagram and summarized below.

FOS Applications:

Sends unformatted events to the DMS. The events are formatted and archived by the DMS at the data server.

MSS:

Sends unformatted events to the DMS. The events are formatted and archived by the DMS at the data server.

Receives MSS related events from the DMS. The DMS uses an events database to determine which events are to be sent to the MSS.

FOS User Interface:

Receives formatted events from the DMS. The user can select the type of events to view.

3.7.2 DMS Event Processing Interfaces

Table 3.7-1. DMS Event Processing Interfaces

Interface Service	Interface Class	Interface Class Description	Service Provider	Service User	Frequency
Event Processing	FdEvEventLogger	Sends events to event handler	DMS	All FOS, MSS	Very frequent

3.7.3 DMS Event Processing Object Model

The FdEventLogger class provides application software a way to send events to the event handler. The user calls the GenEvent operation passing the appropriate parameters whenever an event needs archived and sent to display. The FdEvEventLogger class will create a FoEvEvent class from the calling parameters and send the FoEvEvent class to the FdEvEventHandler class.

The FdEvEventHandler class routes events to the FdEvEventArchiver. The FdEvEventHandler class uses the FdEvEventConfig class to determine which events need to be sent to the FdEvEventArchiver class.

The FdEvEventConfig class contains incoming and outgoing event filters. The user can select the type of events that need to be sent to the event archiver, and the type of events the user station needs to listen for.

The FdEvEventArchiver class receives unformatted events from event handlers. The FdEvEventArchiver uses the FdEvEventDb event database class to determine how to format the events. The formatted events are archived using the FdEvEventFile class, and multicasted over the network to user stations. The FdEventArchiver class also uses the event database to determine if a procedure needs initiated. If a procedure needs initiated the FdEvEventArchiver class will instantiate a FdEvProcedure class.

The FdEvEventListener class listens for formatted events on the network. The FoEvEventListener filters events by using information provided in the FdEvEventConfig class, and then sends the events to display.

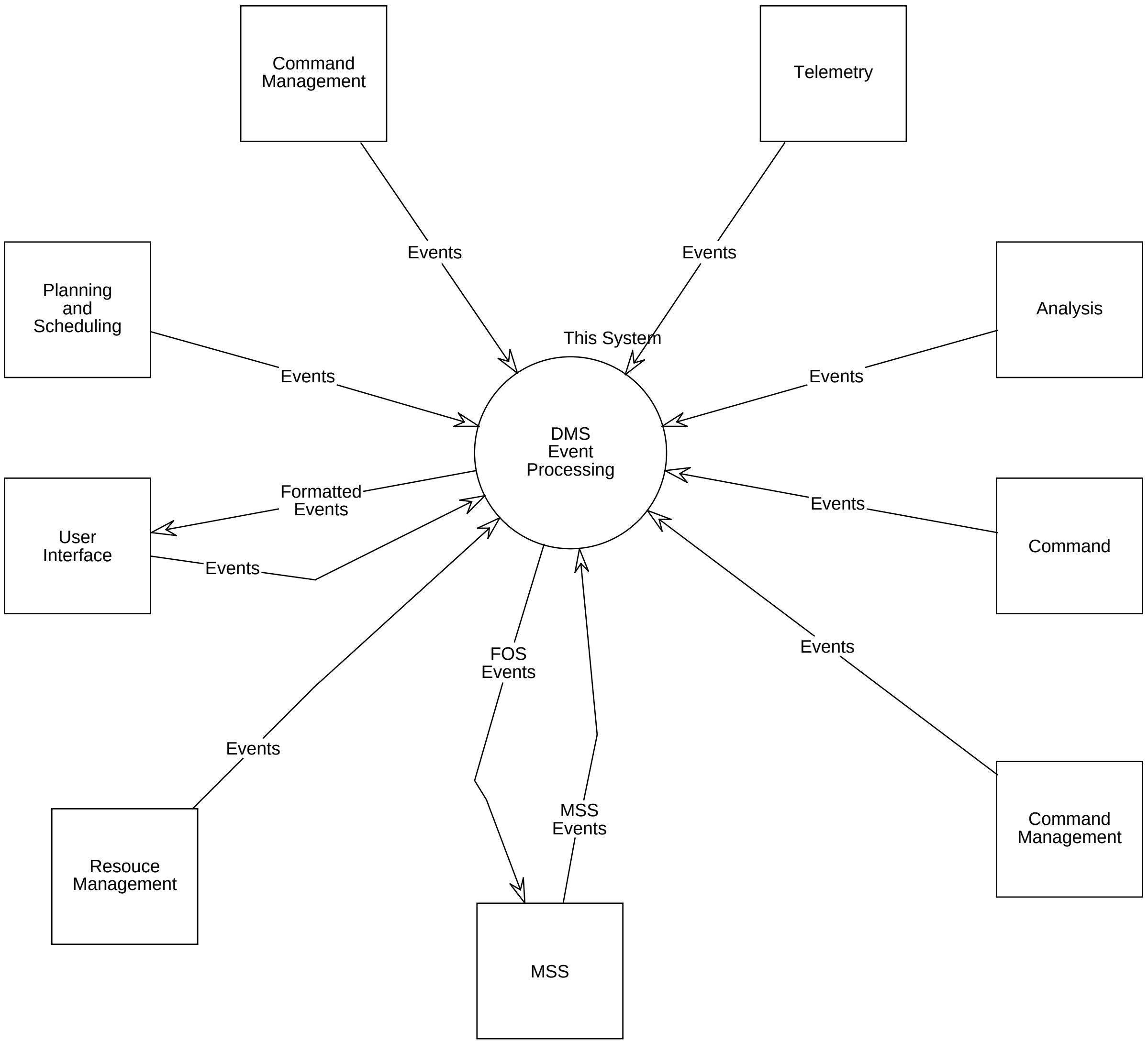


Figure 3.7-1. DMS Event Processing Context

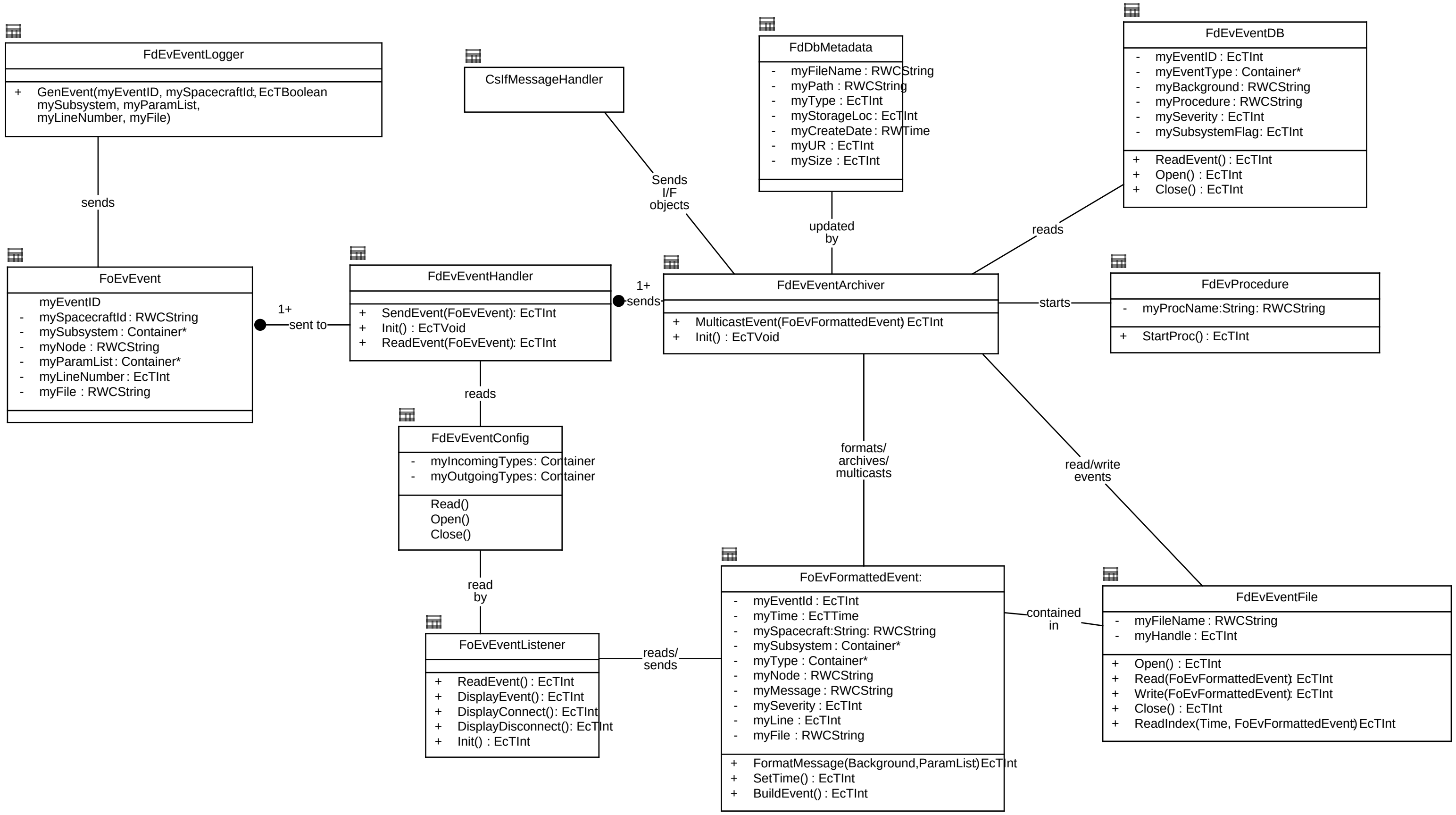


Figure 3.7-2. DMS Event Processing Object Model

3.7.4 DMS Event Processing Dynamic Model

3.7.4.1 DMS Event Processing Scenario Abstract

The purpose of the Event Processing scenario is to describe the process by which events are generated, archived and sent to displays. The event trace for this scenario can be found in Figure-3.7-3.

3.7.4.2 DMS Event Processing Summary Information

Interfaces:

User Interface

Analysis

Telemetry

Command

Resource Management

Real-time Contact Manager

Planning and Scheduling

Command Management

Stimulus:

FdEventLogger genevent operation is called by application software.

Desired Response:

Formatted event is created, archived, and multicasted.

Pre-Conditions:

Event applications initialized.

Post-Conditions:

Event is stored at data server, and displayed at user station.

3.7.4.3 DMS Event Processing Scenario Description

The FdEvEventLogger provides applications with a way to send events to the FdEvEvent Handler. The FdEvEventLogger is responsible for creating a FoEvEvent and sending it to the FdEvEventHandler.

The FdEvEventHandler class sends the FoEvEvent class to the FdEvEventArchiver class. The FdEvEventArchiver class creates an FoEvFormattedEvent by using the FdEvEventDB event database class, and information provided in the FoEvEvent class. The FdEvEventArchiver class will use the event id to index into the FdEvEventDB. The FdEvEventArchiver starts the FdEvProcedure class if the FdEvEventDb classs defines a procedure to initiate. The FdEvEventArchiver archives FoEvFormattedEvent classes to the FdEvEventFile, and then multicasts FoEvFormattedEvent classes over the network.

The FoEvEventListener class reads FoEvFormattedEvent classes off the network. The FoEvEventLister class uses the FdEvEventConfig class to filter events. The FoEvEventListener sends FoEvFormattedEvent classes to the event displays.

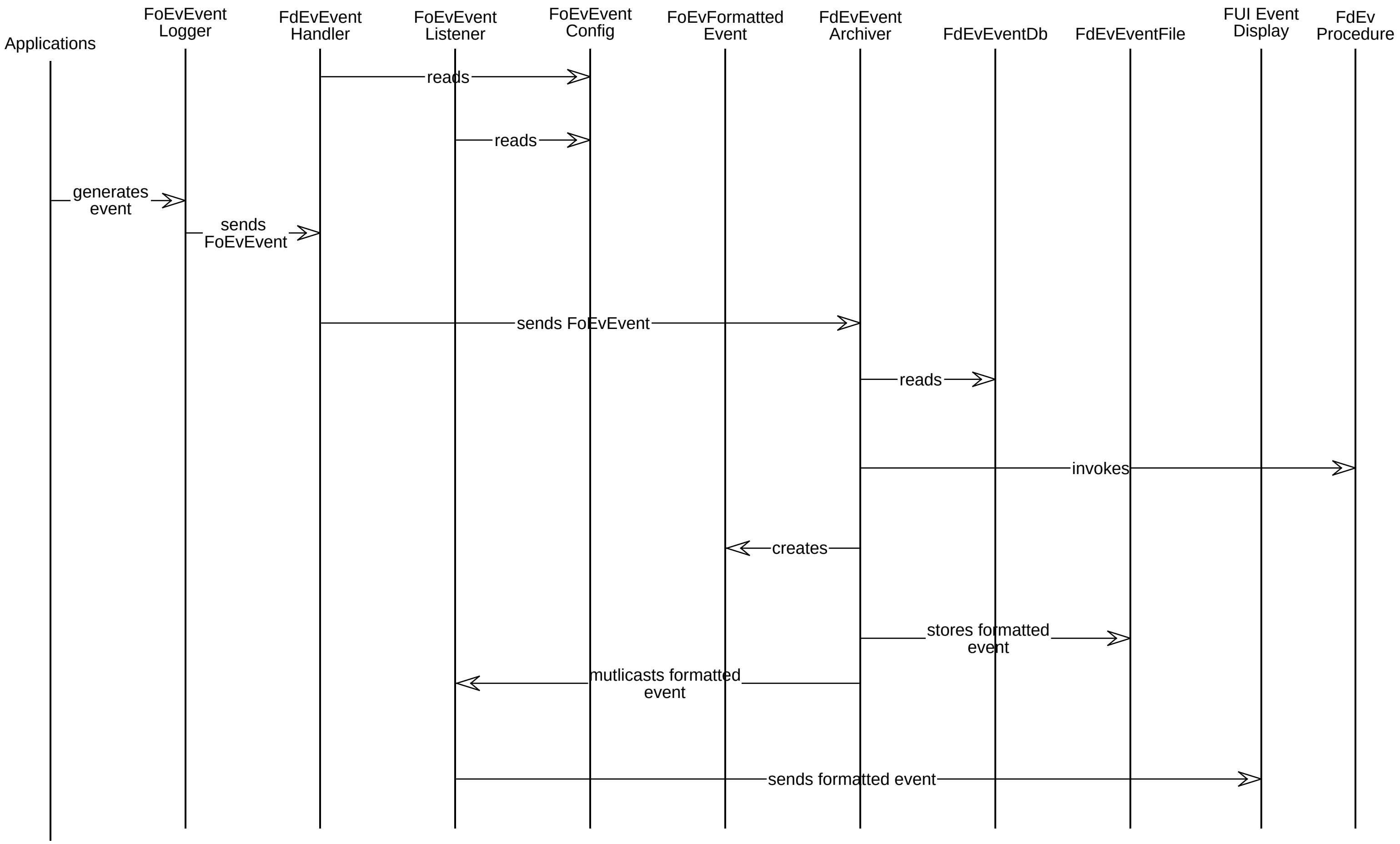


Figure 3.7-3. DMS Event Procesing Event Trace

3.7.3.5 DMS Event Processing Data Dictionary

Class Name: FdEvEventConfig

Description: Configuration file used by the Event Handler and the Event Listener to filter events.

Features:

Attributes:

myIncomingTypes

Description:The types of events the Event Listeners listens for

myOutgoingTypes

Description:The types of events the Event Handler sends to the Event Archiver myTriggerFlag

Description:Turns triggers on/off.

myNodeType

Description:Type of machine (EOC workstation, IST, Data Server, RT Server)

Operations:

FdEventConfig:ReadConfig

Description:Reads in all configuation information

Class Name: FdEvEventDb

Description: Event Database is used to format events, and give information about what to do with events

Features:

Attributes:

myEventID: EcTInt

Description:event number used to index into the event database

myEventType:Container*

Description:type of event

(TLM, CMD, CMS, DMS, FUI, CSMS, RMS)

myBackground:String

Description:background text that is combined with myParamList of FoEvEvent to create a formatted event.

(Printer %s Failed) Printer Failed is background

myProcedure:String

Description:Procedure name that needs to be triggered.

myTriggerLoc:EcTInt

Description:Where a trigger can be initiated
(Workstation, IST, DataServer, RTServer)

mySeverity:EcTInt

Description:Events are warnings or alarms.

mySubsystemFlag:EcTInt

Description:Flag indicating if event should goto subsystem.

Operations:

FdEvEventDb::Open

Description:opens event database

FdEvEventDb::Close

Description:closes event database

FdEvEventDb::ReadEvent

Description:reads record from event database

Class Name: FdEvEventFile

Description: Archived event file.

Features:

Attributes:

myFileName::String

Description:Hourly event filename. Naming convention is
YYYYDDDHH.evt

myHandle::EcTInt

Description:handle of open event file

Operations:

FdEvEventFile:Open

Description:opens event file

FdEvEventFile:Close

Description:closes event file

FdEvEventFile::Read

Description:reads formatted event record from event file

FdEvEventFile::Write

Description:writes formatted event record from event file

Class Name: FdEvLogger

Description: Generated anytime an event needs to be displayed and archived.
Reference FoEvEvent for description of attributes.

Features:

Attributes:

Operations:

FdEvEventLogger:GenEvent(myEventID, mySpacecraftID, mySubsystem, myParamList, myLine, myFile)

Class Name:FdEvProcedure

Description:This class starts up procedures

Features:

Attributes:

myProcName

Description:Name of Procedure to initiate.

Operations:

FdEvProcedure::StartProc

Description:Starts Procedure name myProcName.

Class Name:FoEvEvent

Description:Generated anytime an event needs to be displayed and archived.

Features:

Attributes:

myEventID: EcTInt

Description:event number used to index into the event database

mySpacecraftID:String

Descripton:identifies spacecraft

(AM1)

mySubsystem:Container*

Description:identifies subsystems

(CERES, MOPITT, MISR, ASTER, MODIS)

myNode:String

Description:identifies node name

(EOC Workstation, IST, Data Server, RT Server)

myParamList:Container*

Description:Parameters that fill in the event message.

(Ex. Printer %s Failed) %s is the Param List

myLineNumber:EcTInt

Description:Line number of event

Use Macro __LINE__ for this argument

myFile:String

Description:File name of event

Use Macro __FILE__ for this argument

Operations:

Class Name: FoEvEventArchiver

Description: Event Archiver archives and multicastes formatted events.

Features:

Attributes:

Operations:

FdEvEventArchiver::Init

Description:initiliazes Event Archiver attributes

FdEvEventArchiver::Run

Description:main loop of Event Archiver

FdEvEventArchiver::MulticastEvent

Description:multicasts formatted event

Class Name: FoEvEventHandler

Description: Event Handler receives all events generated on the local workstation or server. The Event Handler uses a configuration file to determine if anevent is sent to the Event Archiver.

Features:

Attributes:

Operations:

FdEvEventHandler::Init

Description:initiliazes Event Handler attributes.

FdEvEventHandler::Run

Description:main loop of Event Handler

FdEvEventHandler::SendEvent

Description:sends unformatted event to the Event Archiver

Class Name: FoEvEventListener

Description: Listens for events on the network, the sends events to FUI Event Analyzer.

Features:

Attributes:

Operations:

FoEvEventListener::ReadEvent

Description:Reads events off the network.

FoEvEventListener::DisplayEvent

Description:Sends event to Event Analyzer.

FoEvEventListener::DisplayConnect

Description:Request from FUI Event Analyzer to send events

FoEvEventListener::DisplayDisconnect

Description:Request from FUI Event Analyzer to quit sending events

FoEvEventListener::Init

Description:Initializes Event Listener attributes

FoEvEventListener::Run

Description:Main loop of Event Listener

Class Name: FoEvFormattedEvent

Description: Event generated by Event Archiver. Event Archiver uses FoEvEvent and the event database to generate the FoEvFormattedEvent. FoEvFormattedEvent gets archived, mutlicasted, and displayed by FUI Event Analyzer.

Features:

Attributes:

myEventID: EcTInt

Description:event number used to index into the event database

myTime:EcTTime

Description:Hourly event filename. Naming convention is YYYYDDDHH.evt

myEventID: EcTInt

Description:event number used to index into the event database

mySpacecraftID:String

Descripton:identifies spacecraft.

mySubsystem:Container*

Description:identifies subsystems (CERES, MOPPITT, MISR, ASTER, MODIS) myType:Container*

Description:type of event

(TLM, CMD, CMS, DMS, FUI, CSMS, RMS)

myNode:String

Description:identifies node name.

(EOC Workstation, IST, Data Server, RT Server)

myMessage:String

Description:The actual event text. Background text from database combined with ParamList from FoEvEvent.

mySeverity:EcTInt

Description:Events are warnings or alarms.

myLineNumber:EcTInt

Description:Line number of event

Use Macro __LINE__ for this argument

myFile:String

Description:File name of event

Use Macro __FILE__ for this argument

Operations:

FoEvFormattedEvent::FormatMessage

Description:Combines background text with ParamList from FoEvEvent

FoEvFormattedEvent::SetTime

Description:Gets system time and sets time attributeof FoEvFormattedEvent.

FoEvFormattedEvent::BuildEvent

Description:Sets attributes within the FoEvFormattedEvent.

3.8 DMS Event Retrieval

A user can build an event request when there is a need to analyze historical events. The event request will consist of start time, stop time, event identifier, event type, subsystem/instrument identifier and spacecraft identifier. The event request is sent to the data server where the requested events are retrieved and sent back to the requesting workstation. The requested events are stored in an event history file. The user interface can display the events contained in the event history file.

3.8.1 DMS Event Retrieval Context

The DMS event retrieval interfaces with the user interface subsystem, as shown in the Context Diagram and summarized below.

FOS User Interface:

Sends unformatted events to the DMS. The events are formatted and archived by the DMS at the data server.

3.8.2 DMS Event Retrieval Interfaces

Table 3.8-1. DMS Event Retrieval Interfaces

Interface Service	Interface Class	Interface Class Description	Service Provider	Service User	Frequency
Event history request	FoRqEventRequest	Used by FUI when requesting event history	DMS	FUI	Frequest

3.8.3 DMS Event Retrieval Object Model

The FoRqEventRequest class provides the user a way to retrieve events from the events archive. The FoRqEventRequest class sends event requests to the FdEvEventRetriever class.

The FdEvEventRetriever class is responsible for reading formatted events from the event archive and creating an event history file from the formatted events.

The FdDsFileManager retrieves event files from long-term storage if needed.

The FdDbMetadata class provides an interface to Sybase. This class allows access to information about all files stored by DMS.

The FdEvFormattedEvent contains information about any event generated by the system. Such information as time, spacecraft id, subsystem, type, node, message severity, event application line, and event application file are contained in this class.

The FdEvEventFile class is a hourly file used to store formatted events.

The FoEvEventHistory class is created from the FdEvEventRetriever class, and is read by the FOS User Interface. This class maintains the formatted events requested by the FOS User Interface.

The FdEvEventRetriever class determines if the event files needed to support the request are online by accessing information provided by the FdDbMetadata class. The FdEventRetriever class makes a request to the FdDsFileManager class if event files are needed from long-term storage. The FdEvEventRetriever class uses information provided in the request to read FoEvFormattedEvent classes from the FdEvEventFile class. A FoEvEventHistory class is created from the FoEvFormattedEvent classes.

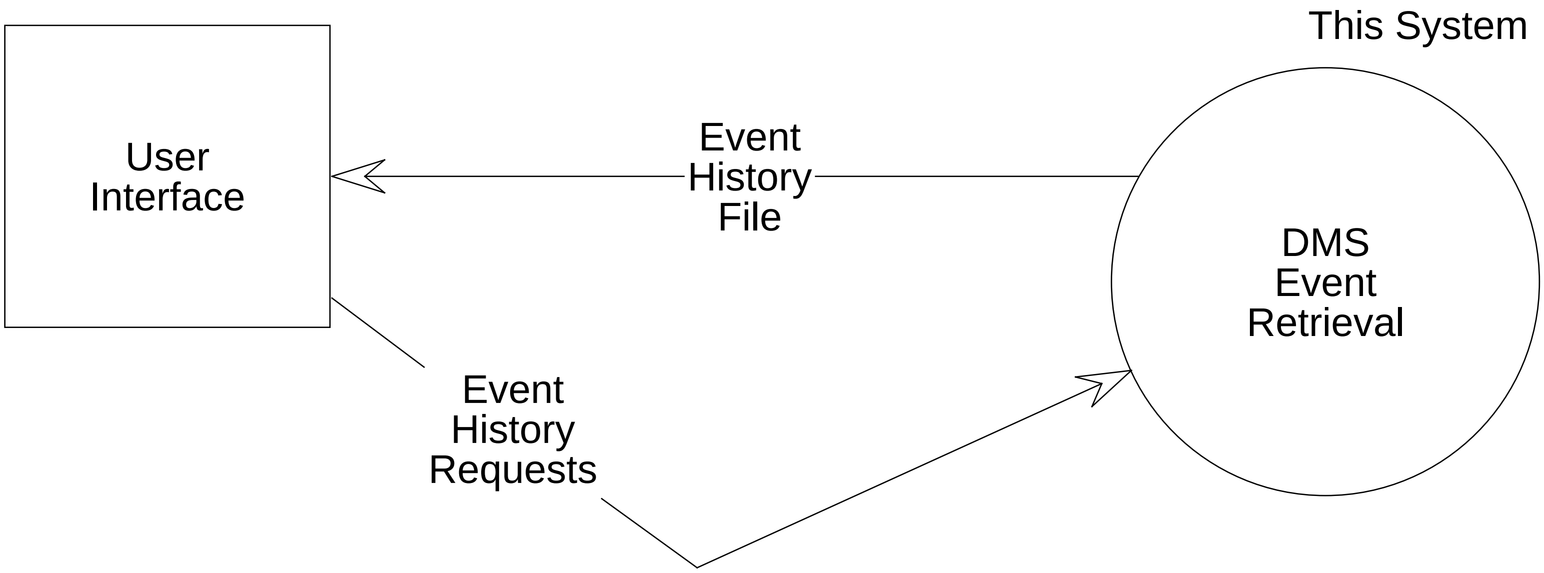


Figure 3.8-1. DMS Event Retrieval Context Diagram

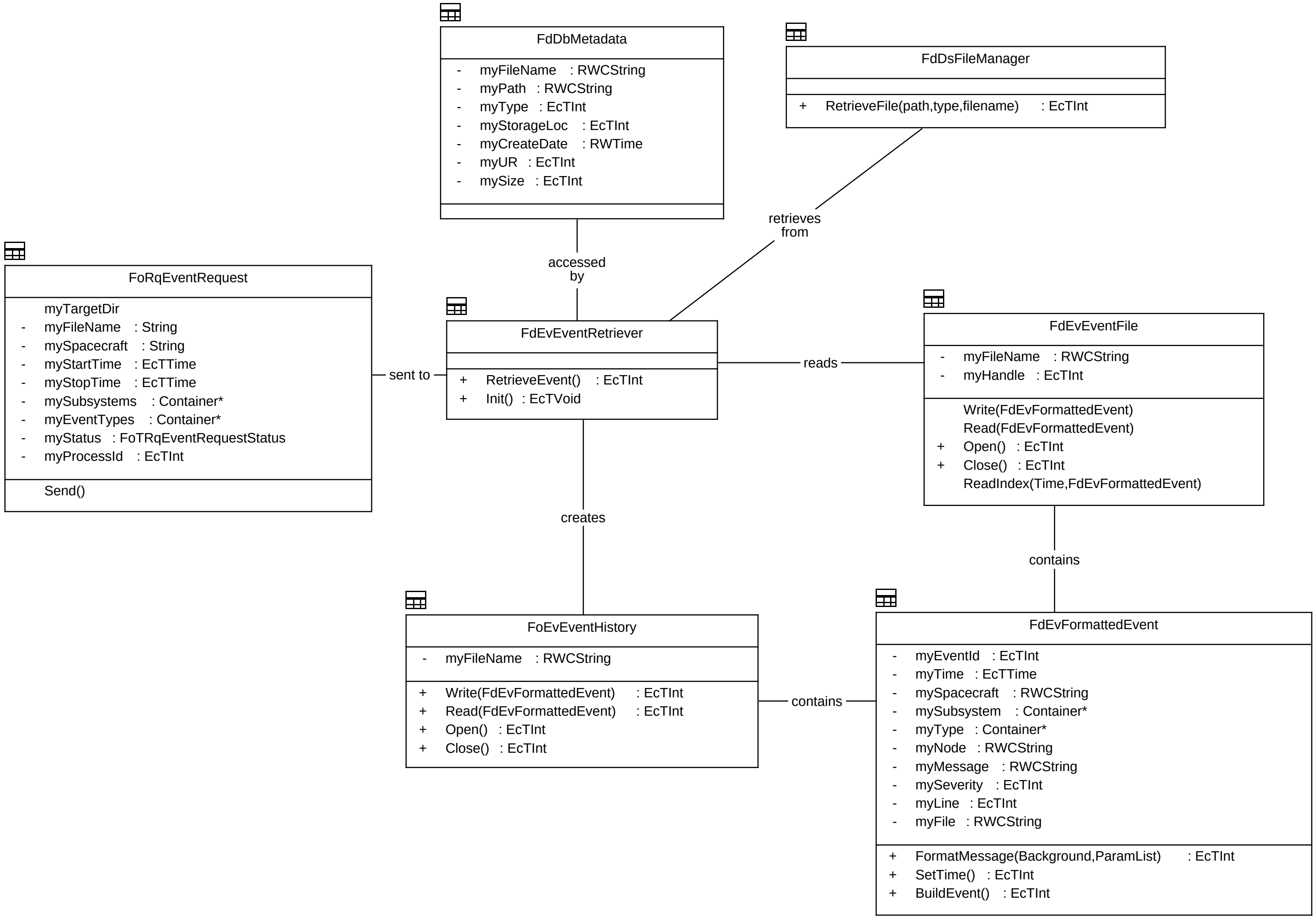


Figure 3.8-2. DMS Event Retrieval Object Model

3.8.4 DMS Event Retrieval Dynamic Model

3.8.4.1 DMS Event Retrieval Scenario Abstract

The purpose of the Event Retrieval scenario is to describe the process by which events are retrieved from the data server, and how an event history file is created. The event trace for this scenario can be found in Figure 3.8-3.

3.8.4.2 DMS Event Retrieval Summary Information

Interfaces:

User Interface

Stimulus:

FoRqEventRequest is instantiated by the FOS User Interface.

Desired Response:

Event History file is created.

Pre-Conditions:

Event applications initialized.

Post-Conditions:

Formatted events are stored in the event history file.

3.8.4.3 DMS Event Retrieval Scenario Description

The FOS User Interface will instantiate an FoRqEventRequest class when an event history file needs to be created. The event request is sent to the FdEvEventRetriever class.

The FdEvEventRetriever class determines if the events file needed to support the request are on-line by accessing information provided by the FdDbMetadata class. The FdEventRetriever class makes a request to the FdDsFileManager class if event files are needed from long-term storage. The FdEvEventRetriever class uses information provided in the request to read FoEvFormattedEvent classes from the FdEvEventFile class. A FoEvEventHistory class is created from the FoEvFormattedEvent classes.

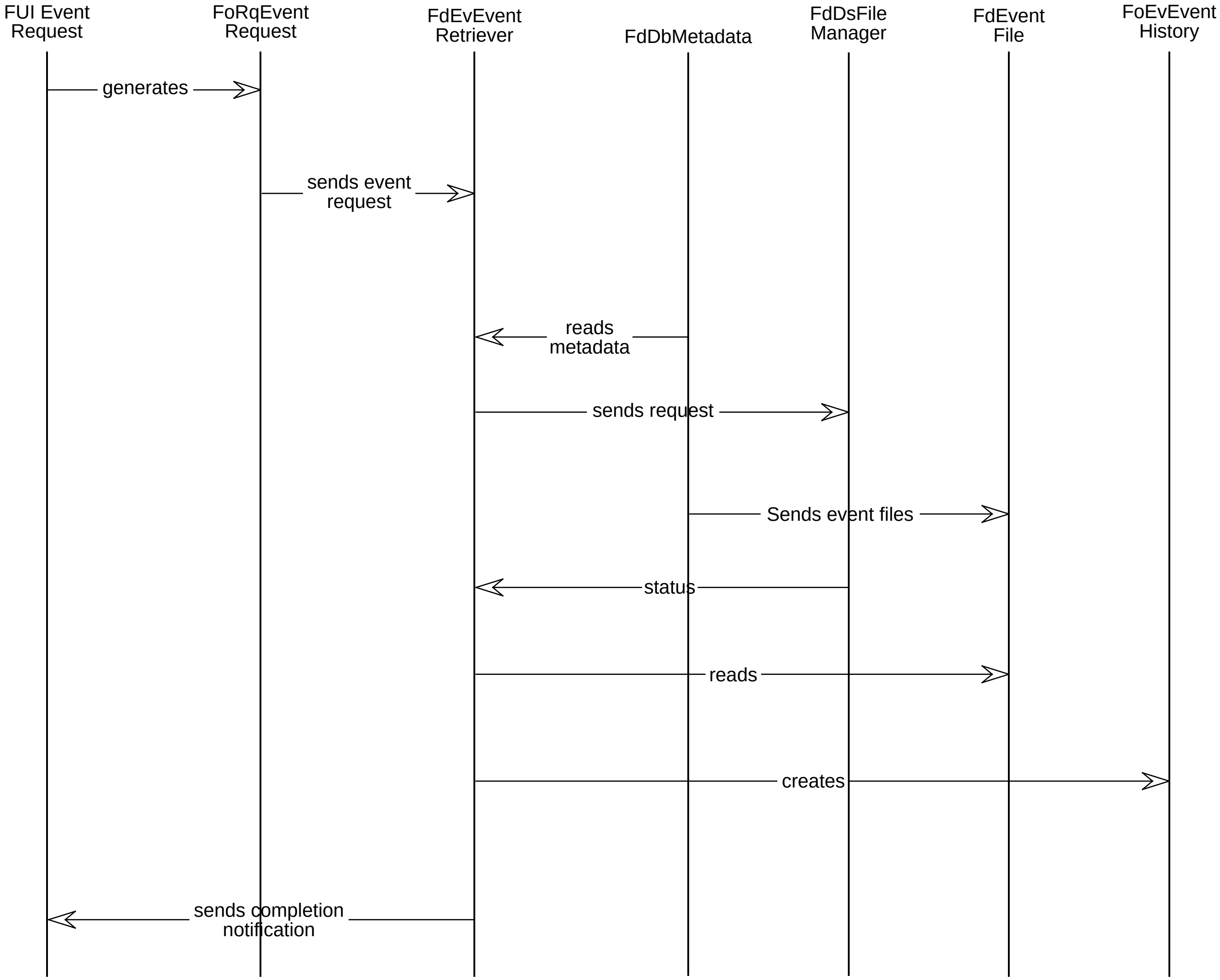


Figure 3.8-3 . DMS Event Retrieval Event Trace

3.8.3.5 DMS Event Retrieval Data Dictionary

Class Name: FdEvEventFile

Description: Archived event file.

Features:

Attributes:

myFileName::String

Description:Hourly event filename. Naming convention is YYYYDDDHH.evt

myHandle::EcTInt

Description:handle of open event file

Operations:

FdEvEventFile:Open

Description:opens event file

FdEvEventFile:Close

Description:closes event file

FdEvEventFile::Read

Description:reads formatted event record from event file

FdEvEventFile::Write

Description:writes formatted event record from event file

Class Name: FdEvEventRetriever

Description: Controller class responsible for building event history files.

Features:

Attributes:

Operations:

FdEvEventRequest:Init

Description:Initializes Event Retriever

FdEvEventRetriever:Run

Description:Main loop of the Event Retriever

Class Name:FdEvFormattedEvent
Description:Event generated by Event Archiver. Event Archiver uses FoEvEvent and the event database to generate the FoEvFormattedEvent. FoEvFormattedEvent gets archived, mutlicasted, and displayed by FUI Event Analyzer.

Features:

Attributes:

myEventID: EcTInt
Description:event number used to index into the event database

myTime:EcTTime
Description:Hourly event filename. Naming convention is YYYYDDDHH.evt

mySpacecraftID:String
Descripton:identifies spacecraft.

mySubsystem:Container*
Description:identifies subsystems (CERES, MOPPITT, MISR, ASTER, MODIS)

myType:Container*
Description:type of event (TLM, CMD, CMS, DMS, FUI, CSMS, RMS)

myNode:String
Description:identifies node name. (EOC Workstation, IST, Data Server, RT Server)

myMessage:String
Description:The actual event text. Background text from database combined with ParamList from FoEvEvent.

mySeverity:EcTInt
Description:Events are warnings or alarms.

myLineNumber:EcTInt
Description:Line number of event
Use Macro __LINE__ for this argument

myFile:String
Description:File name of event
Use Macro __FILE__ for this argument

Operations:

FoEvFormattedEvent::FormatMessage
Description:Combines background text with ParamList from FoEvEvent

FoEvFormattedEvent::SetTime
Description:Gets system time and sets time attributeof FoEvFormattedEvent.

FoEvFormattedEvent::BuildEvent
Description:Sets attributes within the FoEvFormattedEvent.

Class Name:FoEvEventHistory
Description:File generated from an event history request. Contains FdEvFormattedEvent(s).

Features:

Attributes:

Operations:

FoEvEventHistory:Open

Description:opens event history file

FoEvEventHistory:Close

Description:closes event history file

FoEvEventHistory::Read

Description:reads formatted event record from event history file

FoEvEventHistory::Write

Description:writes formatted event record from event history file

Class Name:FoRqEventRequest

Description:Interface class with between DMS and FUI Event Analyzer. This class is used to request event history from DMS.

Features:

Attributes:

myTargetDir:String

Description:Directory where event history file is created.

myFileName:String

Descripton:Event History Filename.

mySpacecraft:String

Descripton:identifies spacecraft .

myStartTime:EcTTime

Description:start time of the event request

myStoptime:EcTTime

Description:stop time of the event request

mySubsystem:Container*

Description:identifies subsystems
(CERES, MOPITT, MISR, ASTER, MODIS)

myType:Container*

Description:type of events requested
(TLM, CMD, CMS, DMS, FUI, CSMS, RMS)

myStatus:EcTInt

Description:Status of event request returned to FUI

myProcessId:EcTInt

Description:Id of the requesting FUI Event Analyzer

Operations:

FoRqEventRequest:Send

Description:Sends event request to Event Retriever

3.9 DMS File Management, External Interfaces, Database Access

The DMS is responsible for providing file management, external interface, and database access utilities. File management utilities allow the user to store, retrieve, and access DMS managed data files. External interface utilities provide access to EDOS back orbit telemetry data, FDF products, and SCDO long term storage data. Database utilities allow the user to update, extract, and retrieve information from Sybase.

3.9.1 DMS File Management, External Interfaces, Database Access Context

The DMS utilities interface with several FOS subsystems and external interfaces, as shown in the Context Diagram and summarized below.

FOS Applications:

Send unix data files and database updates to the DMS for storage.

Receives unix data files, database information, and external file arrival notifications from the DMS.

EDOS:

Sends back orbit telemetry file, which is then merged with real-time telemetry to create a seamless archive.

FDF:

Sends FDF data products to the EOC. The DMS notifies subsystems of the arrival of FDF data.

SCDO:

Receives data files from the EOC. Data files are stored at the DACC for the life of the mission.

Sends long term archive files to the DMS. The DMS requests data files from SCDO on an as needed basis.

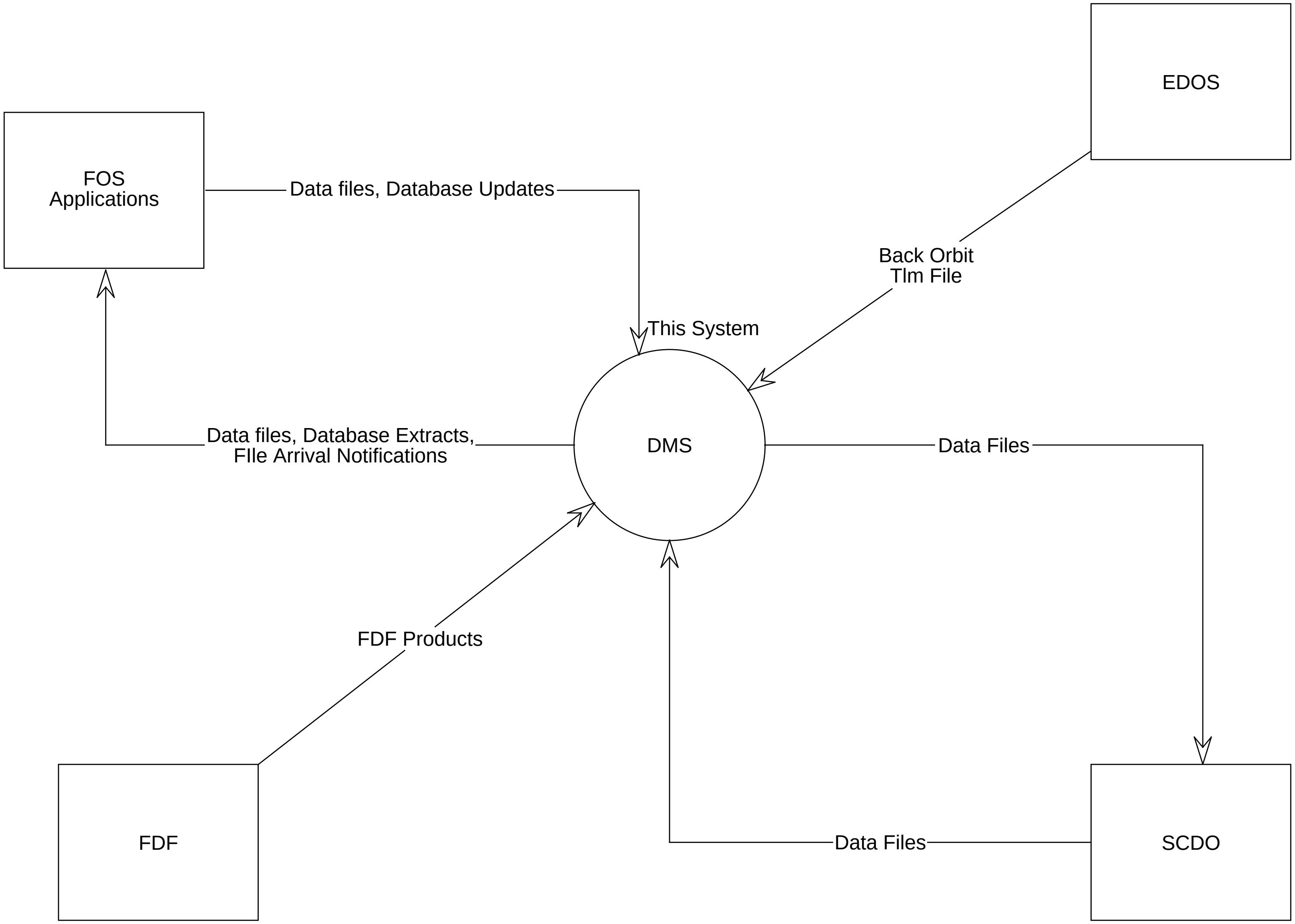


Figure 3.9-1. DMS File Management, External Interfaces, Database Access Context Diagram

3.9.2 DMS File Management, External Interfaces, Database Access Interfaces

Table 3.9-1. DMS File Management, External Interfaces, Database Access Interfaces

Interface Service	Interface Class	Interface Class Description	Service Provider	Service User	Frequency
File Access	FdDsFileAccessor	Allows for storage and retrieval of data files	DMS	All FOS	Frequent
Database Access	FdDbDBAccessor	Allows for extracting, updating, adding, and deleting from Sybase database tables	DMS	PAS, FUI, CMS, CMND	Frequent

3.9.3 DMS File Management, External Interfaces, Database Access Object Model

The FoDsFile class provides a wrapper for all subsystems to use when opening, reading, writing, and closing data files, and the FdDsFileAccesser class provides a mechanism for storing and retrieving FoDsFile classes. Software applications will link FoDsFile and FdDsFileAccessor into their executable. Applications can obtain information about data files by using the GetFileInfo operation within the FdDsFileAccesser class. Such information as path, type, creation date, size, and long term storage can be retrieved. FdDsFileAccessor is a DMS owned proxy that communicates with the FdDsFileManager by sending and retrieving FdDsFileInformation.

The FdDsFileManager class is responsible for maintaining FOS data files. The FdDsFileManager uses the FdDbFileMeta to add, delete, update, and get file information from Sybase. Data files are stored at the EOC local archive for a minimum of 7 days, and some as long as a month. When files are created they are sent to the GSFC DACC for long term storage.

FdDsFileManager will send data files to the DACC via the FdLtIngest class. When a file is successfully archived by the DACC a Universal Reference (UR) identification is returned. The FdDsFileManager updates file metadata with the UR identifier. The EOC can acquire any file that has been stored at the DACC by using the UR identifier. The FdDsFileManager uses the FdLtDataServer class to retrieve any data file needed from the DACC.

The FdDsDiskCleaner class is responsible for purging data files from the EOC local disk. The FdDsDiskCleaner class use information from the FdDsFileConfig class as to how long data files are to remain at the EOC local archive.

The FdDsExternalInterface class is responsible for determining when data is received from EDOS, FDF, or an IST. The DMS sends a FoNtNotification class to users of the data. The FoNtNotification class informs users of the filename and path of the data received.

The FoDbAccessor provides application software an interface to Sybase. Application software uses the FoDbAccessor to connect and disconnect from Sybase, and then the software uses the appropriate subclass to add, delete, update, and get information.

The DMS uses the FdDbFileMeta, FdDbTlmMeta, and FdDbOdbTable to access information about data files, telemetry archive, and operational databases. The FoDbCatalogEntry allows CMS to store and retrieve information about loads. The FdDbActivityDef, FdDbActCmd, and FdDbActCmdParm interfaces are used by Planning and Scheduling and CMS. These tables are used to retrieve information about activities.

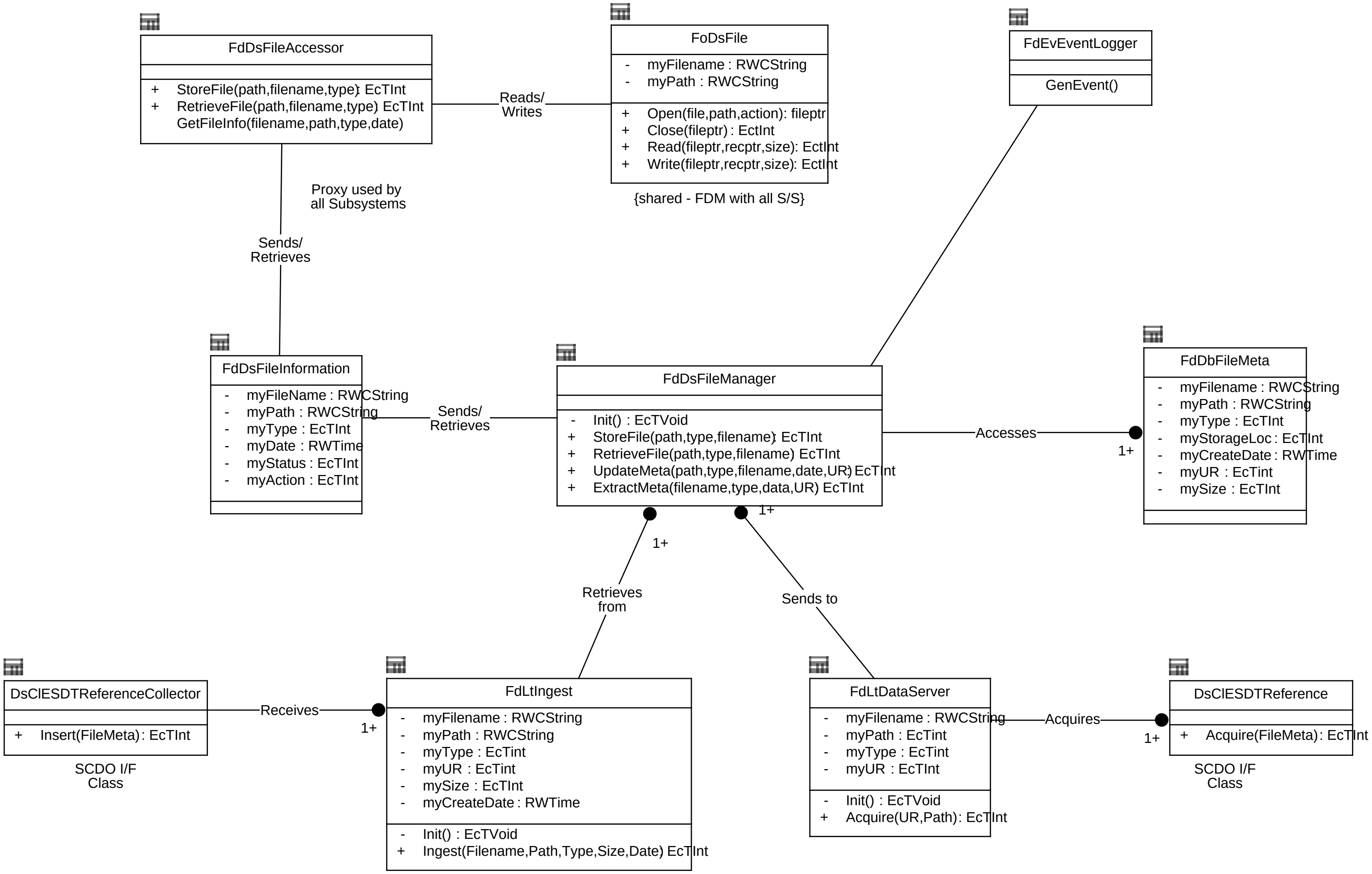


Figure 3.9-2. DMS File Management, External Interfaces, Database Access Object Model

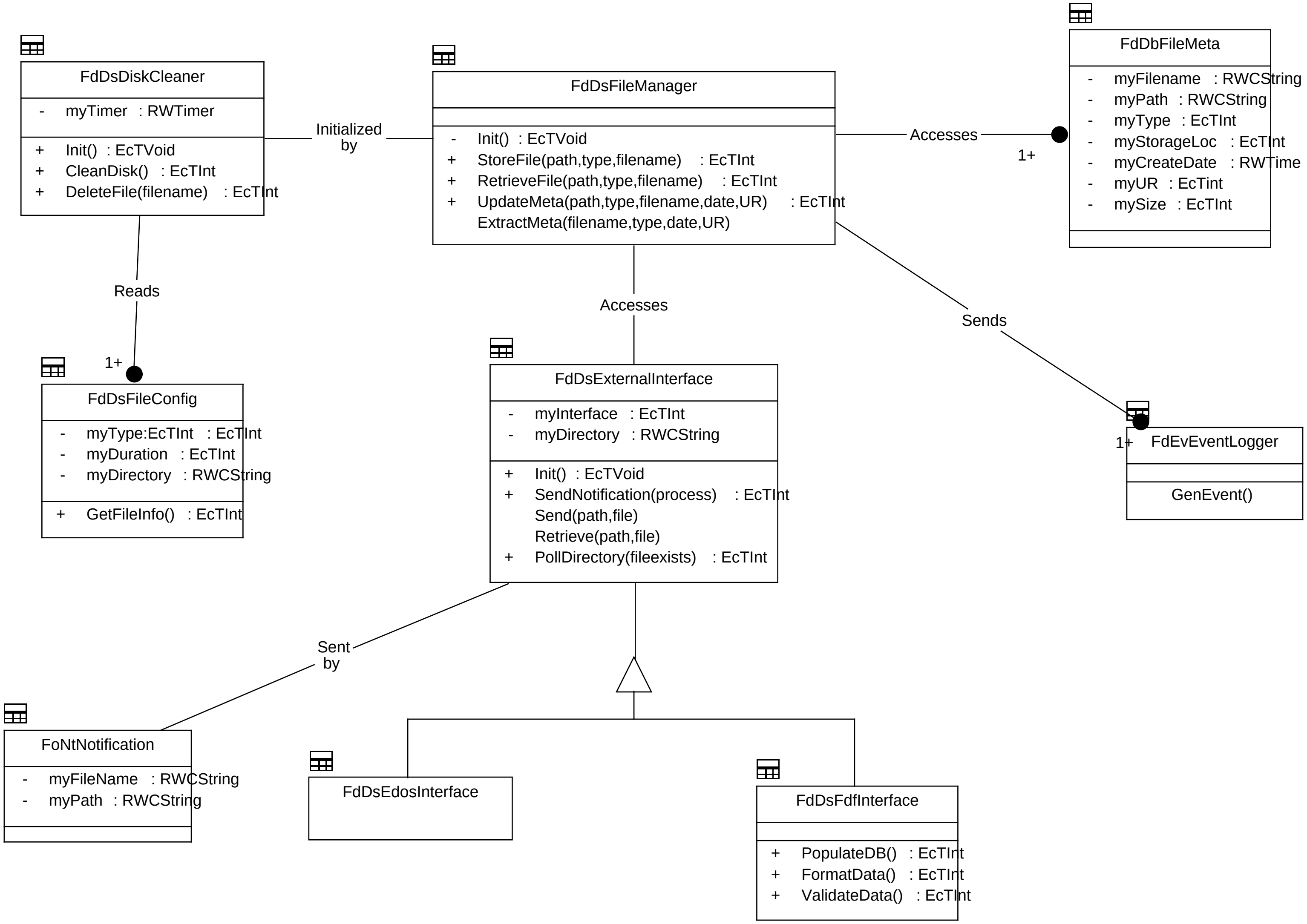


Figure 3.9-3.DMS File Management, External Interfaces, Database Access Object Model

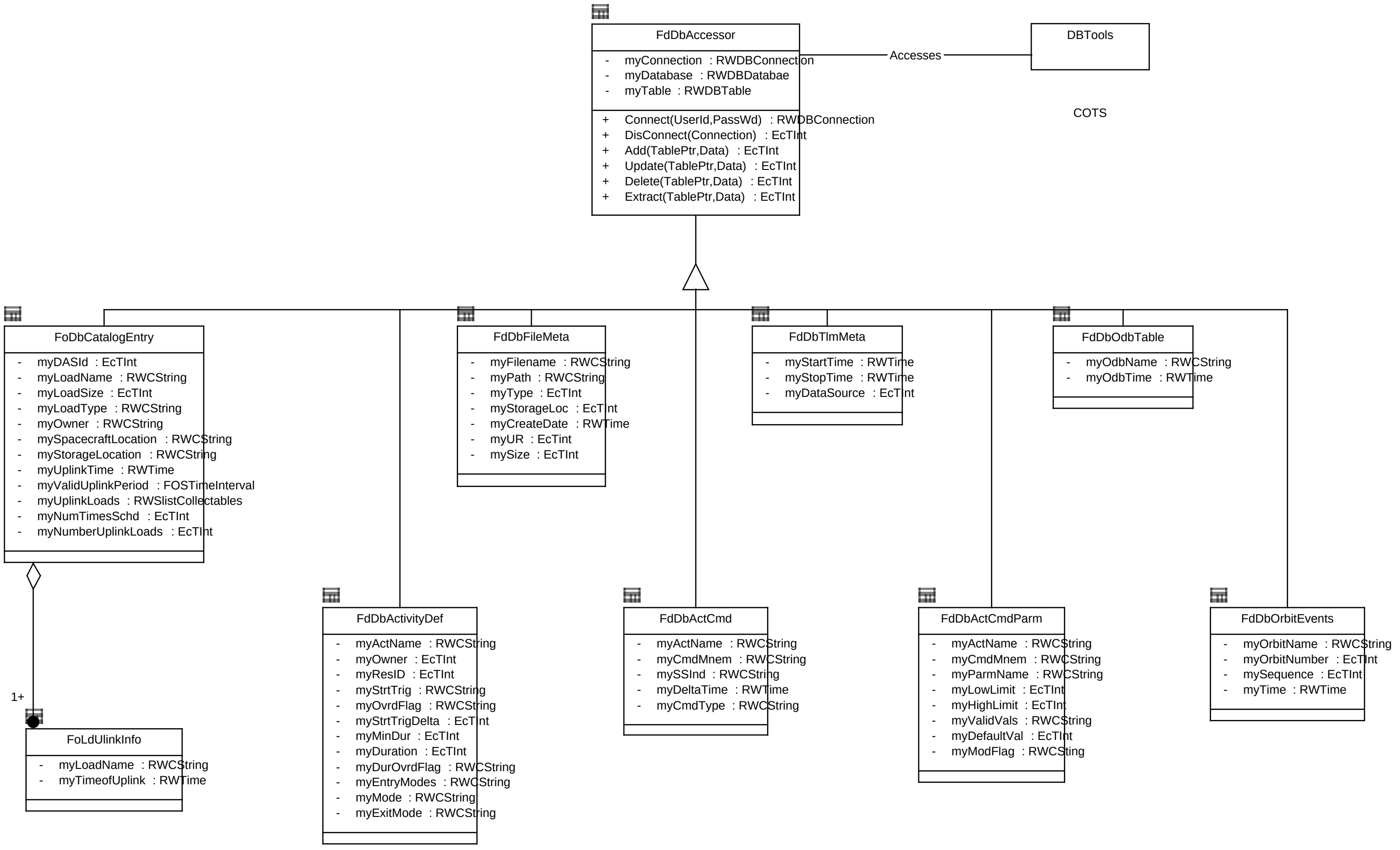


Figure 3.9-4.DMS File Management, External Interfaces, Database Access Object Model

3.9.4 DMS File Management, External Interfaces, Database Access Dynamic Model

3.9.4.1.1 DMS File Storage Scenario Abstract

The purpose of the File Storage scenario is to describe the process by which unix data files are stored at the EOC and sent to long term storage. The event trace for this scenario can be found in Figure 3.9-5.

3.9.4.1. 2 DMS File Storage Summary Information

Interfaces:

User Interface

Analysis

Telemetry

Command

Resource Management

Real-time Contact Manager

Planning and Scheduling

Command Management

Stimulus:

FdDsFileAccessor receives a request to store a data file.

Desired Response:

Successful status returned to FOS application.

Participating Classes:

FoDsFile

FdDsFileAccessor

FdDsFileManager

FdDbFileMeta

FdLtIngest

DsClESDTReferenceCollector

Pre-Conditions

SCDO interface established.

Post-Conditions

Data file is stored at EOC local archive, and at the SCDO long term archive.

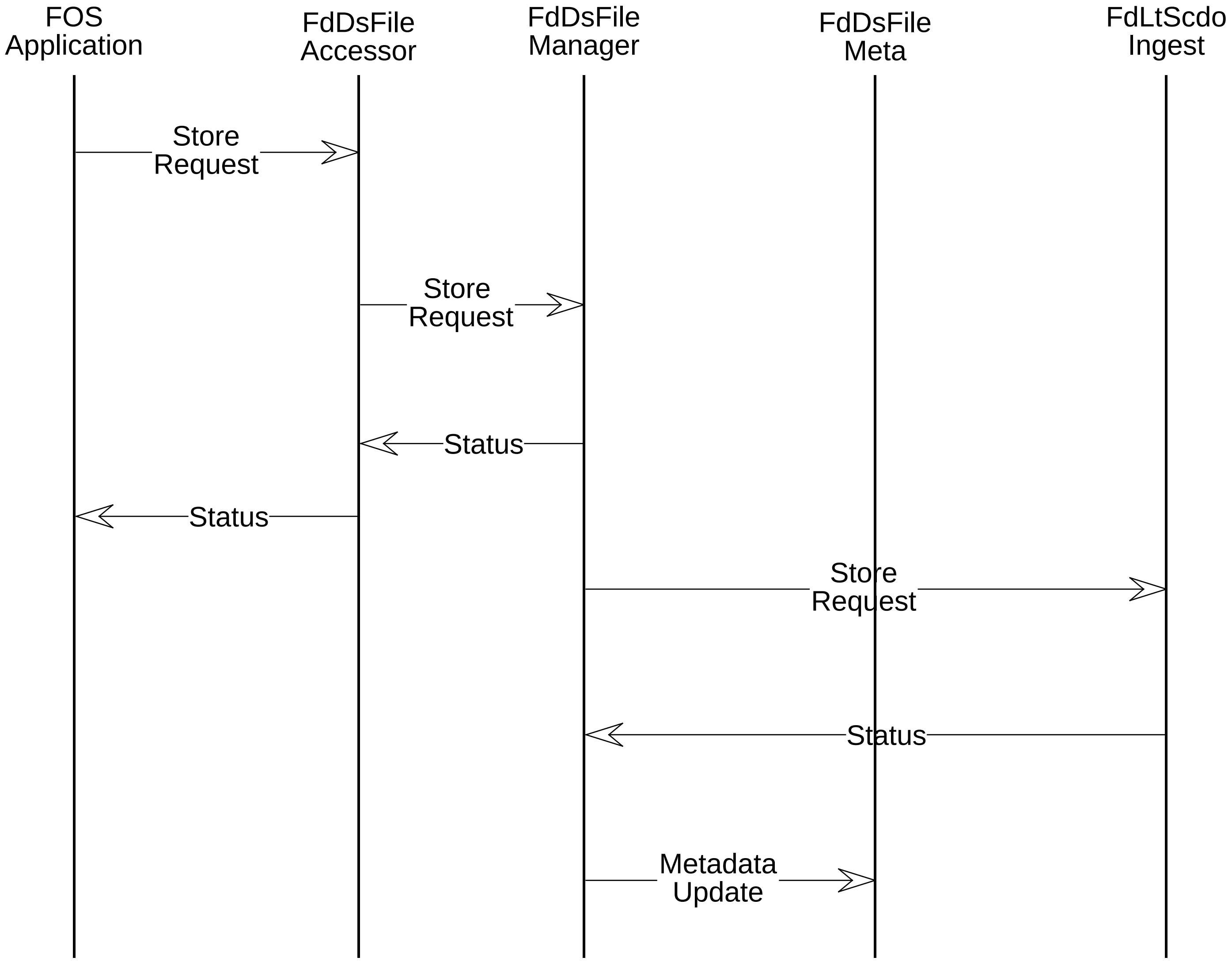


Figure 3.9-5. DMS File Storage Event Trace

3.9.4.3 DMS File Storage Scenario Description

FOS applications will store data files by using the FdDsFileAccessor class. The FdDsFileAccessor will send FdDsFileInformation to the FdDsFileManager concerning the data file that needs stored. The FdDsFileManager will copy the data file to the appropriate directory by using information from the FdDsFileInformation class. File metadata within Sybase is updated using the FdDsFileMeta class. FdDsFileMeta is a subclass of the FdDbAccessor class. The FdDsFileManager class will send the data file to long term storage by calling the FdLtIngest Ingest operation. FdLtIngest class uses SCDO provided interface classes to store data files in long term storage.

3.9.4.2.1 DMS File Retrieval Scenario Abstract

The purpose of the File Retrieval scenario is to describe the process by which unix data files are retrieved from the EOC local archive, or SCDO long term archive. The event trace for this scenario can be found in Figure 3.9-6.

3.9.4.2.2 DMS File Retrieval Summary Information

Interfaces:

User Interface

Analysis

Telemetry

Command

Resource Management

Planning and Scheduling

Command Management

Stimulus:

FdDsFileAccessor receives a request to retrieve a data file.

Desired Response:

FOS application receives successful status and pointer to unix data file.

Participating Classes:

FoDsFile

FdDsFileAccessor

FdDsFileManager

FdDbFileMeta

FdLtDataServer

DsClESDTReference

Pre-Conditions

SCDO interface established.

Post-Conditions

Data file is retrieved from either EOC local archive, or SCDO long term archive.

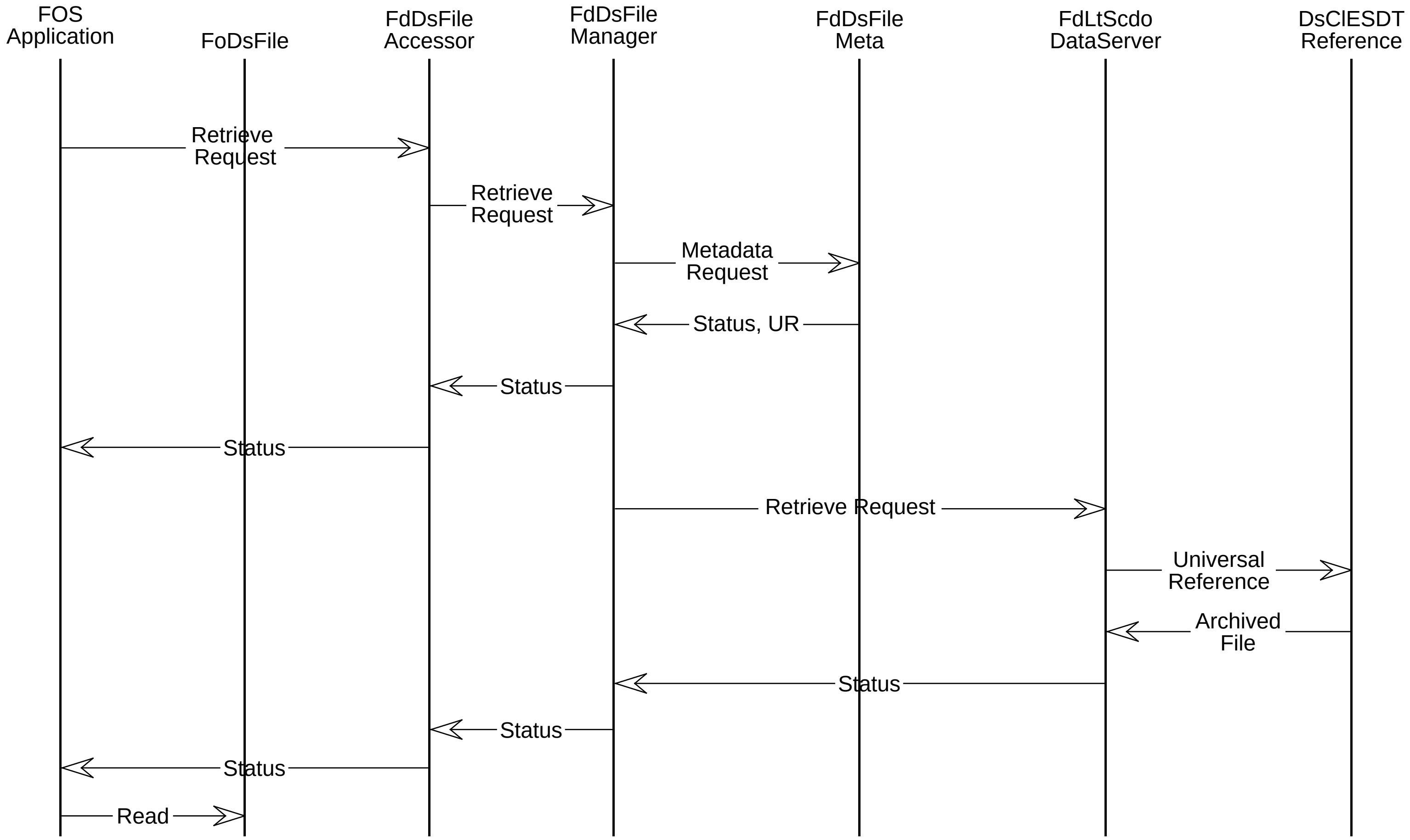


Figure 3.9-6. DMS File Retrieval Event Trace

3.9.4.2.3 DMS File Retrieval Scenario Description

FOS applications will retrieve data files by using the FdDsFileAccessor class. The FdDsFileAccessor will send FdDsFileInformation to the FdDsFileManager concerning the data file that needs retrieved. The FdDsFileManager will access file metadata via the FdDsFileMeta class. The FdDsFileManager will used the file metadata to determine if the data file needs retrieved from long term storage. If so, a request is sent to the FdLtDataServer class for the data file. The FdLtDataServer will request files from SCDO provided classes. Once the data file is retrieved from long term storage, the FdDsFileManager will copy the data file to the requested path. The FOS application will then use the FoDsFile class to open, close, read, and write to a file.

3.9.4.3.1 DMS Sybase Table Access Scenario Abstract

The purpose of the Sybase Table Access scenario is to describe the process by which a FOS application can update or extract table information from Sybase. The event trace for this scenario can be found in Figure 3.9-7.

3.9.4.3.2 DMS Sybase Table Access Summary Information

Interfaces:

User Interface

Planning and Scheduling

Command Management

Command

Stimulus:

FoDbAccessor receives an update or extract request from a FOS application.

Desired Response:

The requesting FOS application updates or extracts information from
within Sybase.

Participating Classes:

FoDbAccessor

Subclass of FoDbAccessor

FOS application Sybase user

DBTools classes (COTS)

Pre-Conditions

Sybase must be initialized.

Post-Conditions

FOS application receives status from DMS DBAccessor

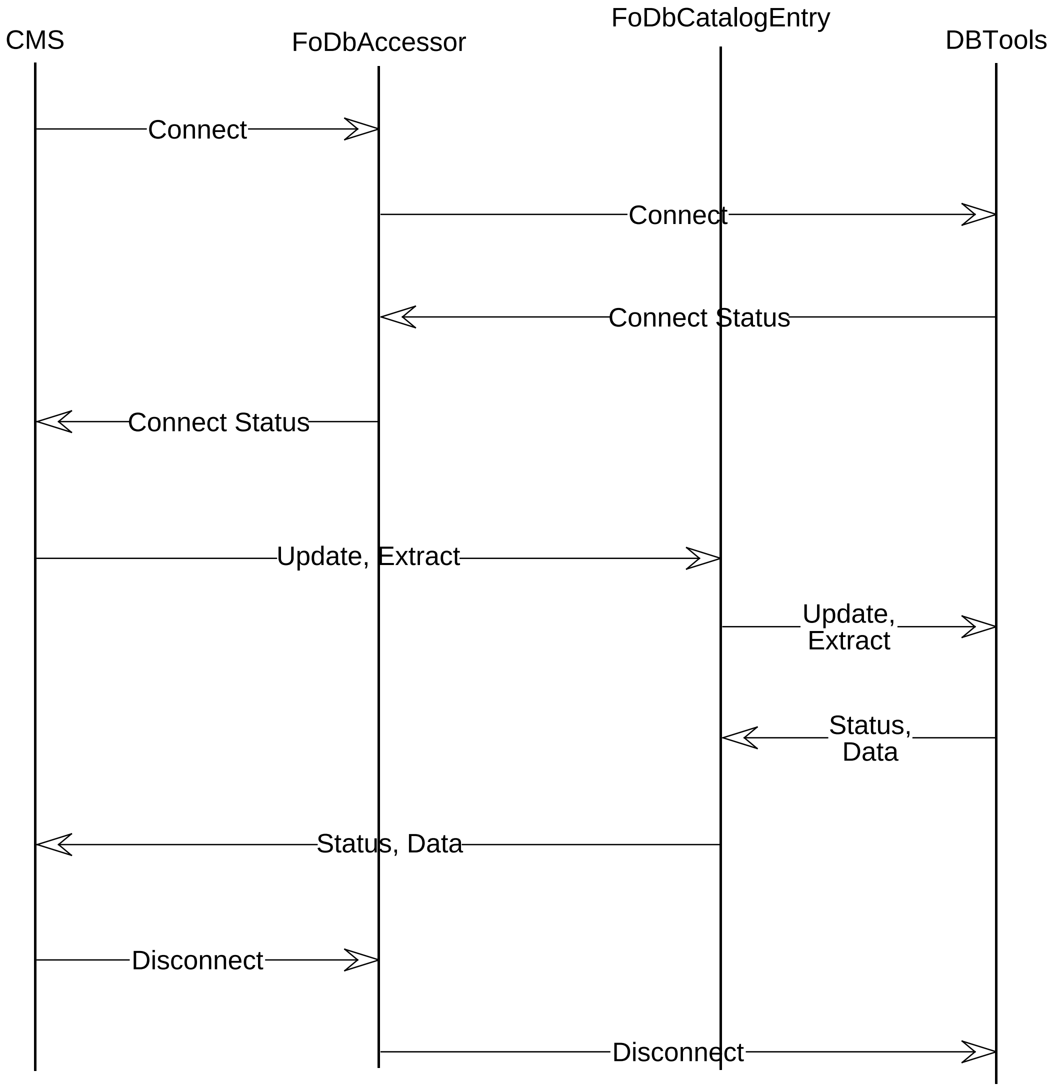


__Figure 3.9-7. DMS Sybase Table Access Event Trace__

3.9.4.3.3 DMS Sybase Table Access Scenario Description

When FOS application software needs to make updates to or extract information from a Sybase table, the application will do so by using the FoDbAccessor class. The FoDbAccessor is a superclass which contains connect, disconnect, update, and extract operations. FOS application software will link in the FoDbAccessor and the subclass which corresponds to the needed database table when they create their executable. The DBAccessor uses DBTools, which is a RoqueWave COTS product, to access Sybase. To update a table, the application software will instantiate a FoDbAccessor subclass, and then call the FoDbAccessor update operation passing needed information. To extract from a Sybase table, the application software will instantiate a FoDbAccessor subclasss, and then call the FoDbAccessor extract operation passing needed information. The FoDbAccessor subclass will contain the retrieved information when the operation completes.

3.9.4.4.1 DMS FDF Interface Scenario Abstract

The purpose of the FDF Interface scenario is to describe the process by which the DMS receives FDF data, and notifies FOS applications that data has arrived. The event trace for this scenario can be found in Figure 3.9-8.

3.9.4.4.2 DMS FDF Interface Summary Information

Interfaces:

Planning and Scheduling

Analysis

Command Management

Stimulus:

FdDsFdfInterface receives FDF products.

Desired Response:

Notification is sent to FOS processes which desire FDF products.

Participating Classes:

FdDsFdfInterface

FdDsExternalInterface

FoNtNotification

FdDsFileManager

Pre-Conditions

FdDsFdfInterface must be polling directory in which FDF products arrives.

Post-Conditions

FdDsFdfInterface polls FDF products directory.

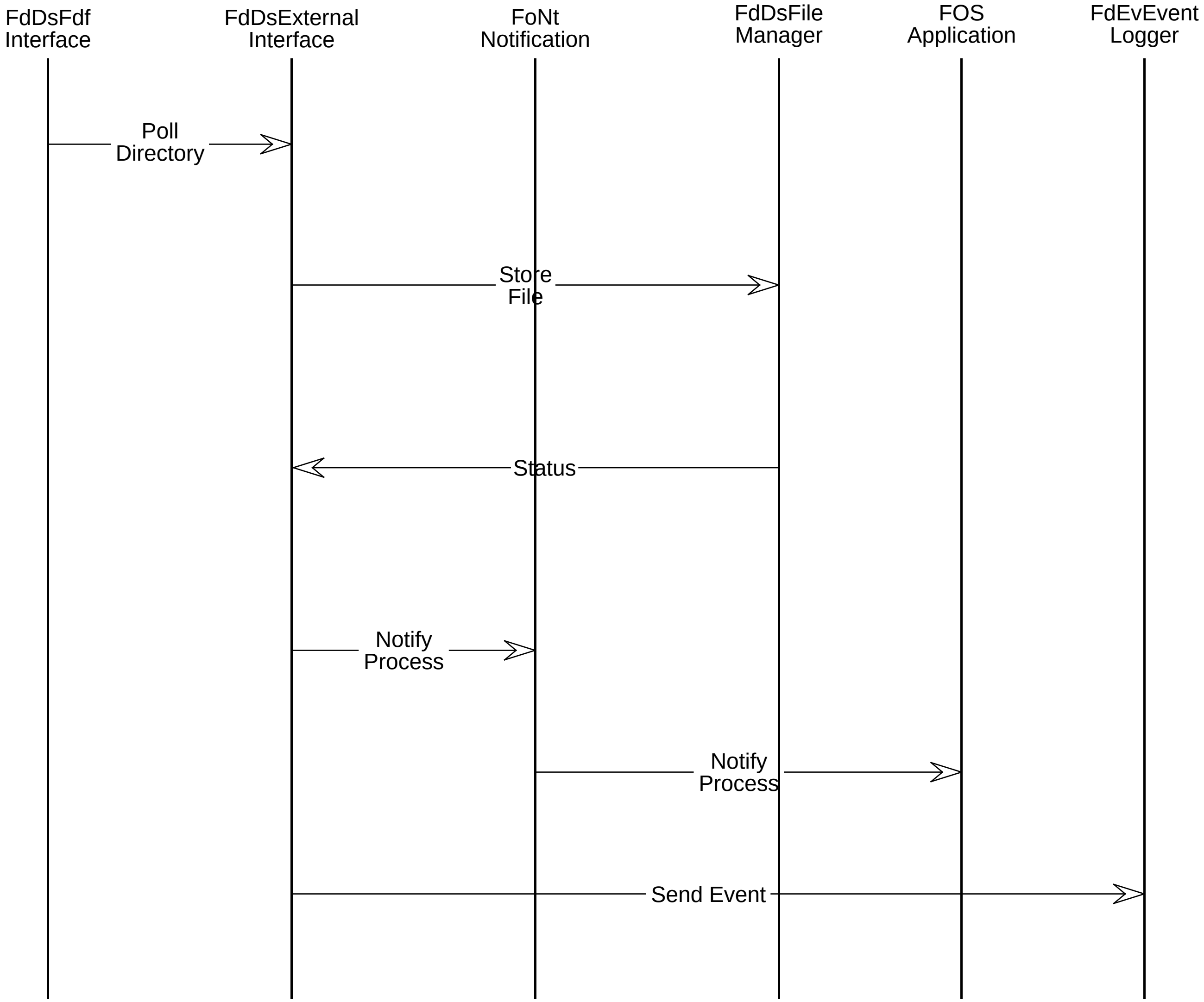


Figure 3.9-8. DMS FDF Interface Event Trace

3.9.4.4.3 DMS FDF Interface Scenario Description

The FdDsFdfInterface class is always polling a dedicated FDF product directory. Once FDF sends data to the dedicated directory, the FdDsFdfInterface class will validate and format the new FDF data. Once validated, the FdDsFdInterface class will store the FDF data using the FdDsFileManager class. Once the data is stored the FdDsFdfInterface class will send FoNtNotifications to Planning and Scheduling, Command Management, and Analysis Subsystems.

3.9.6 DMS File Management, External Interfaces, Database Access Data Dictionary

FdDbAccessor

class **FdDbAccessor**

This class is used to interface with Sybase. The user connect to Sybase, disconnect from Sybase, update table information, and extract table information. Extract and update calls might need to be moved to the subclass level.

Public Functions

RWDBConnection **Connect**(UserId, PassWd)

This function allows a user to connect to a Sybase Database.

EcTInt **DisConnect**(Connection)

Disconnect

This function allows a user to disconnect from Sybase

EcTInt **Extract**(TablePtr, Data)

This member function extracts data from a Sybase table. THis is a generic extract, and this function may evolve to multiple types of extracts.

EcTInt **Update**(TablePtr, Data)

This member function updates a Sybase table. This is a generic update, and this function may evolve to multiple types of updates

Private Data

RWDBConnection **myConnection**

This member variable is the connection to Sybase.

RWDBDatabae **myDatabase**

This member variable is the database the myConnection points to.

RWDBTable **myTable**

This member variable is the table the myConnection points to.

FdDbActCmd

class **FdDbActCmd**

The Activity Command Table provides the defintions of commands that make up a specific activity

Base Classes

public **FdDbAccessor**

Private Data

RWCString **myActName**

activity name specifies a unique identifier for a given activity.

RWCString **myCmdMnem**

command mnemonic represents the mnemonic of an ATC stored command, or a valid EOS Command Language (ECL) directive

RWCString **myCmdType**

command type represents the type of command used in the activity.

EcTInt **myDeltaTime**

delta time is the time offset from the start time or the stop time of the activity.

RWCString **mySSind**

start/stop time indicator is used to specify whether the delta time specified for the command is associated with the start time or the stop time of the activity.

FdDbActCmdParm

class **FdDbActCmdParm**

Base Classes

public **FdDbAccessor**

Private Data

RWCString **myActName**

activity name specifies a unique identifier for a given activity.

RWCString **myCmdMnem**

parameter name identifies the parameter associated with a command.

EcTInt **myDefaultVal**

default value indicates the value to be used if the no value is specified when the activity is scheduled.

EcTInt **myHighLimit**

high limit indicates the highest value in the range of values for the parameter.

EcTInt **myLowLimit**

low limit indicates the lowest value in the range of values for the parameter.

RWCString **myModeFlag**

modifiable flag indicates whether a different parameter value can be specified when scheduling an activity.

RWCString **myValidVals**

valid values indicate the discrete values in which the parameter must occur.

FdDbActivityDef

class **FdDbActivityDef**

stp/omt class definition 1457203

Base Classes

public **FdDbAccessor**

Private Data

RWCString **myActName**

activity name specifies a unique identifier for a given activity.

RWCString **myDurOvrdFlag**

duration override flag is used to indicate if the duration specified for activity may be overridden when scheduling an activity

EcTInt **myDuration**

duration specifies the default duration for the activity.

RWCString **myEntryModes**

entry modes specifies all valid modes of the resource at the time the activity is scheduled.

RWCString **myExitMode**

string mode specifies the mode of the resource at the end of the activity.

ECTInt **myMinDur**

minimum duration specifies the minimum duration for the activity.

RWCString **myMode**

mode specifes the mode of the resource during the activity period.

RWCString **myOvrdFlag**

start trigger override flag is used to indicate if the start trigger event may be overridden when scheduling an activity.

EctInt **myOwner**

owner specifies the user ID of the person/group who is authorized to define, modify, and/or schedule the give activity.

EcTInt **myResID**

resource ID specifies the name of the resource that the activity operates on.

RWCString **myStrtTrig**

start trigger specifies the name of the event that is used to schedule the give activity.

EcTInt **myStrtTrigDelta**

start trigger delta indicates the time offset, in seconds, from the start trigger event used for sheduling the activity.

FdDbFileMeta

class **FdDbFileMeta**

This class maintains information about data files The user can access this information through the DBAccessor.

Base Classes

public **FdDbAccessor**

Private Data

RWTime **myCreateDate**

file creation date

RWCString **myFilename**

name of file

RWCString **myPath**

path where file is located

EcTInt **mySize**

size of data file

EcTInt **myStorageLoc**

storage location (i.e, local EOC archive, long-term, or both)

EcTInt **myType**

Type of file (i.e., report, archive,event,etc...)

EcTint **myUR**

Universal Reference - this applies when file is sent to long term storage

FdDbOdbTable

class **FdDbOdbTable**

THe ODB table class is used to determine when databases went on-line

Base Classes

public **FdDbAccessor**

Private Data

RWCString **myOdbName**

Operational database name

RWTime **myOdbTime**

Time ODB went on-line

FdDbOrbitEvents

class **FdDbOrbitEvents**

This class allows access to the orbit event table. This table is populated when the EOC receives FDF data.

Base Classes

public **FdDbAccessor**

Private Data

RWCString **myOrbitName**

Orbit Event Name

EcTInt **myOrbitNumber**

Acutal Orbit Number

EcTInt **mySequence**

Sequence of event in a given orbit

RWTime **myTime**

Time of an Orbit

FdDbTlmMeta

class **FdDbTlmMeta**

Table that contains information about telemetry data that is stored at the EOC and the DACC

Base Classes

public **FdDbAccessor**

Private Data

EcTInt **myDataSource**

SOurce of telemetry R/T, Back Orbit, NCC, EDOS

RWTime **myStartTime**

Start time that corresponds to a stop time of tlm data

RWTime **myStopTime**

Stop time that corresponds to a start time of tlm data

FdDsDiskCleaner

class **FdDsDiskCleaner**

This class cleans the EOC local archive by removing old files from the disk. The old files can be retrieved from long term archive if needed.

Public Functions

EcTInt **CleanDisk**(void)

This member function wakes up daily and cleans old data from the EOC local archive.

EcTInt **DeleteFile**(filename)

THis function removes file from the EOC local archive.

EcTVoid **Init**(void)

This member function initializes variables and connnections.

Private Data

RWTimer **myTimer**

This member variable is a daily timer.

FdDsEdosInterface

class **FdDsEdosInterface**

This class is derived from FdDsExternalInterface. This class is responsible for polling a directory waiting for back orbit telemetry.

Base Classes

public **FdDsExternalInterface**

FdDsExternalInterface

class **FdDsExternalInterface**

THis class is a base class that allows for sending, receiving external data, polls directorys for data, and sends notifications when data arrives.

Public Functions

EcTVoid **Init**(void)

Initializes variables and connections

EcTInt **PollDirectory**(fileexists)

Polls a directory waiting for data to arrive from external interace. Notifies subsystems data when data arives.

EcTInt **Retrieve**(file)

Retrieves a file from an external interface

EcTInt **Send**(file)

Sends a file to an external interface

EcTInt **SendNotification**(process)

Notifies subsystems when external data arrives.

Private Data

RWCString **myDirectory**

Directory data files are stored in

EcTInt **myInterface**

External interface connection - may not be one when using polling directory interface

FdDsFdfInterface

class **FdDsFdfInterface**

This class is derived from FdDsExternalInterface. This class provides utilities for formating, validating, and populating Sybase with FDF data.

Base Classes

public **FdDsExternalInterface**

Public Functions

EcTInt **FormatData**()

Formats FDF in a format usable by FOS appliations

EcTInt **PopulateDB**()

Populates Sybase with FDF data

EcTInt **ValidateData**()

Validates FDF data

FdDsFileAccessor

class **FdDsFileAccessor**

Public Functions

EcTInt **GetFileInfo**(filename, path)

THis member function get information about a DMS managed data file

EctInt **RetrieveFile**(path, filename, type)

This member function retrieves a DMS managed file

EcTInt **StoreFile**(path, filename, type)

This member function stores a data file with DMS

FdDsFileConfig

class **FdDsFileConfig**

This class contains information about the data files in the EOC local archive.

Public Functions

EcTInt **GetFileInfo**()

This member function reads data file information from a config file.

Private Data

RWCSTring **myDirectory**

This member variable is the directory where data file types reside.

EcTInt **myDuration**

This member variable holds the length of time a given data file type remains at the local EOC archive.

EcTInt **myType**

This member variable is the type of data file.

FdDsFileInformation

class **FdDsFileInformation**

This class gets passed between the FdDsFileManager and the FdDsFileAccessor classes. It contains information for storing and retrieving files, and updating and extracting file metadata.

Private Data

EcTInt **myAction**

This member variable contains action to be taken. Store, Retrieve, UpdateMeta, or ExtractMeta.

RWTime **myDate**

This member variable contains creation data of file.

RWCString **myFileName**

This member variable contains the actual filename.

RWCString **myPath**

This member variable contains the path of the file.

EcTInt **myStatus**

This member variable contains status of the request.

EcTInt **myType**

This member varialbe contains file type.

FdDsFileManager

class **FdDsFileManager**

This class stores and retrieves files, updates file metadata, and extracts file metadata.

Public Functions

EcTInt **ExtractMeta**(filename, type, date, UR)

Extracts data file information from Sybase.

EcTInt **RetrieveFile**(path, type, filename)

THis member function retrieves a file from the DMS managed area.

EcTInt **StoreFile**(path, type, filename)

This member function takes a file from path and stores it in DMS managed directory.

EcTInt **UpdateMeta**(path, type, fileanme, date, UR)

Updates information about data files in Sybase

FdLtDataServer

class **FdLtDataServer**

This class provides a interface with the SCDO Data Server.   This class retrieves data files needed from long term storage.

Public Functions

EcTInt **Acquire**(UR, Path)

This member function acquire data from long term storage by passing the Universal Refernece.

Private Functions

EcTVoid **Init**(void)

This member functions initializes variables and connections

Private Data

RWCString **myFilename**

This member variable contains name of file to retrieve.

EcTint **myPath**

This member variable contains path to put long term file in.

EcTint **myType**

This member variable contains type of file to retrieve.

EcTInt **myUR**

This member variable contains the Universal Reference of file to retrieve.

FdLtIngest

class **FdLtIngest**

This class is the interface class used to send data files to long term storage

Public Functions

EcTInt **Ingest**(Filename, Path, Type, Size, Date)

This member function sends a local archive file to long term storage

Private Functions

EcTVoid **Init**(void)

This member function initializes variables and connections.

Private Data

RWTime **myCreateDate**

This member variable is the file creation date. .

RWCString **myFilename**

This member variable contains name of file sent to long term storage.

RWCString **myPath**

This member variable contains path where SCDO Ingest pulls file from.

EcTInt **mySize**

This member variable contains size of file

EcTint **myType**

This member variable contains type of file to send to long term storage.

EcTint **myUR**

This member variable is the Universal Reference of the file - returned from SCDO Ingest

FoDbCatalogEntry

class **FoDbCatalogEntry**

This class provide CMS with interface with Catalog Entry table within Sybase. Information about loads is accessed using this class.

Base Classes

public **FdDbAccessor**

Private Data

EcTInt **myDASId**
EcTInt **myNumTimesSchd**
EcTInt **myNumberUplinkLoads**
RWSlistCollectables **myUplinkLoads**

THe id of the DAS in which the load was requested to be uplinked.

RWCString **myLoadName**

The unique name identifying the load.

EcTInt **myLoadSize**

The size of the load in words.

RWCString **myLoadType**

The type of load - table, RTS, ATC, flight software, or microprocessor

RWCString **myOwner**

The user or group that owns the load.

RWCString **mySpacecraftLocation**

The location of the load in the spacecrafts memory.

RWCString **myStorageLocation**

The location of the load in DMS.

RWTime **myUplinkTime**

The time at which the load was uplinked to the spacecraft.

FOSTimeInterval **myValidUplinkPeriod**

The period of time for which the load is valid.

FoDsFile

class **FoDsFile**

Class used by FOS to access data files. This class gives FOS a generic interface to data files. This class will evolve to have many reads and writes of files.

Public Functions

EctInt **Close**(fileptr)

This classes closes a previously opened file.

fileptr **Open**(file, path, action)

This member functions opens the specified data file.

EctInt **Read**(fileptr, recptr, size)

This member function reads the file pointed to by fileptr.

EctInt **Write**(fileptr, recptr, size)

This member funciton writes to file pointed to by fileptr.

Private Data

RWCString **myFilename**

Name of file to open, close, read, or write to.

RWCString **myPath**

Path of filename being accessed.

FoLdUlinkInfo

class **FoLdUlinkInfo**

Private Data

RWCString **myLoadName**

Name of Load

RWTime **myTimeofUplink**

Time load was uplinked to spacecraft

FoNtNotification

class **FoNtNotification**

This class is used to notify processes that a data file is available for processing.

Private Data

RWCString **myFileName**

This member variable contains name of file available for processing.

RWCString **myPath**

This member varaible contains path of available file.

3.10 DMS Telemetry Archiver

3.10.1 DMS Telemetry Archiver Context

The DMS Telemetry Archiver interfaces are described below and displayed in the context diagram.

RMS:

Initializes this instance of the archiver and provides the data required for this archiver to configure itself properly.

Telemetry:

Sends a telemetry or dump EDU to be archived.

File Archival:

Receives the hourly telemetry file or the dump file.

3.10.2 DMS Telemetry Archiver Interfaces

Table 3.10.2 DMS Telemetry Archiver Interface

Interface Service	Interface Class	Interface Class Description	Service Provider	Service User	Frequency
Provide data units from TLM to DMS archiver.	FdArTlmArchProxy	Pass telemetry data units to DMS.	DMS	TLM	Frequently
	FdArEDU	Data unit container			
Get configuration information from RMS.	FdCfRMSConfigProxy	Pass information to DMS.	RMS	DMS	Upon startup

Note: Above table is subject to change.

3.10.3 DMS Telemetry Archiver Object Model

The FdArTlmArchiver object is configured according to information received from RMS via the FdCfRMSConfigProxy class. Once initialized, FdArTlmArchiver receives EDUs via FdArTlmArchProxy and manages their storage. FdArTlmArchiver receives the EDU by invoking the retrieveData function of the FdArUserData object. FdArUserData encompasses the interface between the archiver and TLM. FdArTlmArchiver then builds and stores the SAU by using the build and store methods contained in the FdArSAU object. FdArTlmArchiver continues receiving EDUs until a data dropout is detected. During this process, FdArTlmArchiver invokes methods contained in FdArSAU to check for valid sequence counts in the data (except for dumps). If sequence gaps are detected, FdArTlmArchiver uses FdMtRTUpdateNotification to update the metadata table for available telemetry.

A single instance of FdArSAU remains persistent throughout the contact. This object is responsible for building and storing the SAU. It also maintains the current telemetry sequence count and verifies that the EDU sequence count is in order. In addition, FdArSAU updates and maintains the start and stop time of the current, contiguous, telemetry stream (this is not done for dumps).

The FdArSAU object stores the SAU. The SAU consists of FdArHeader and FdArUserData. FdArHeader contains the information pertinent to the particular EDU. FdArUserData contains the actual EDU and the operation required to retrieve the EDU from TLM.

The FdArHourlyTlmFile object is responsible for operations on the local archive file. The FdMtUpdateNotification object provides the interface between the archiver and the archive metadata table. This notification is sent whenever a new file is opened. The FdMtRTUpdateNotification object provides the interface to the 'available telemetry' metadata table. This table contains start and stop times of all contiguous telemetry data in the archive.

3.10.4 DMS Telemetry Archival Dynamic Model

3.10.4.1 DMS Telemetry Archival Scenario Abstract

3.10.4.2 DMS Telemetry Archival Summary Information

Interfaces:

RMS

TLM

File Archival

Stimulus:

Receipt of telemetry packets from telemetry.

Desired Response:

Storage of Standard Archive Units (SAUs) to a file.

Pre-Conditions:

Archiver software has been initiated.

Post-Conditions:

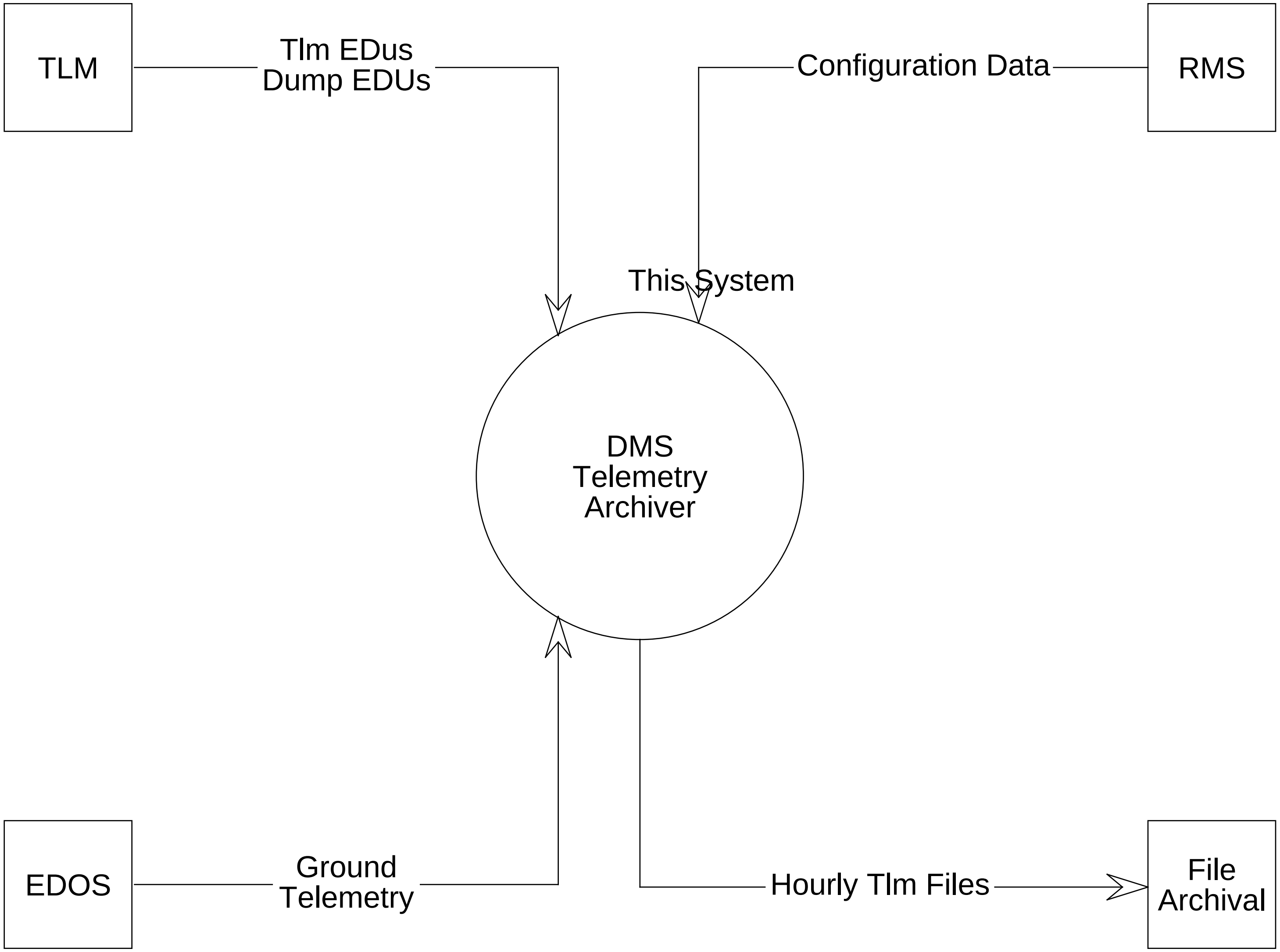


Figure 3.10-1. DMS Telemetry Archiver Context Diagram

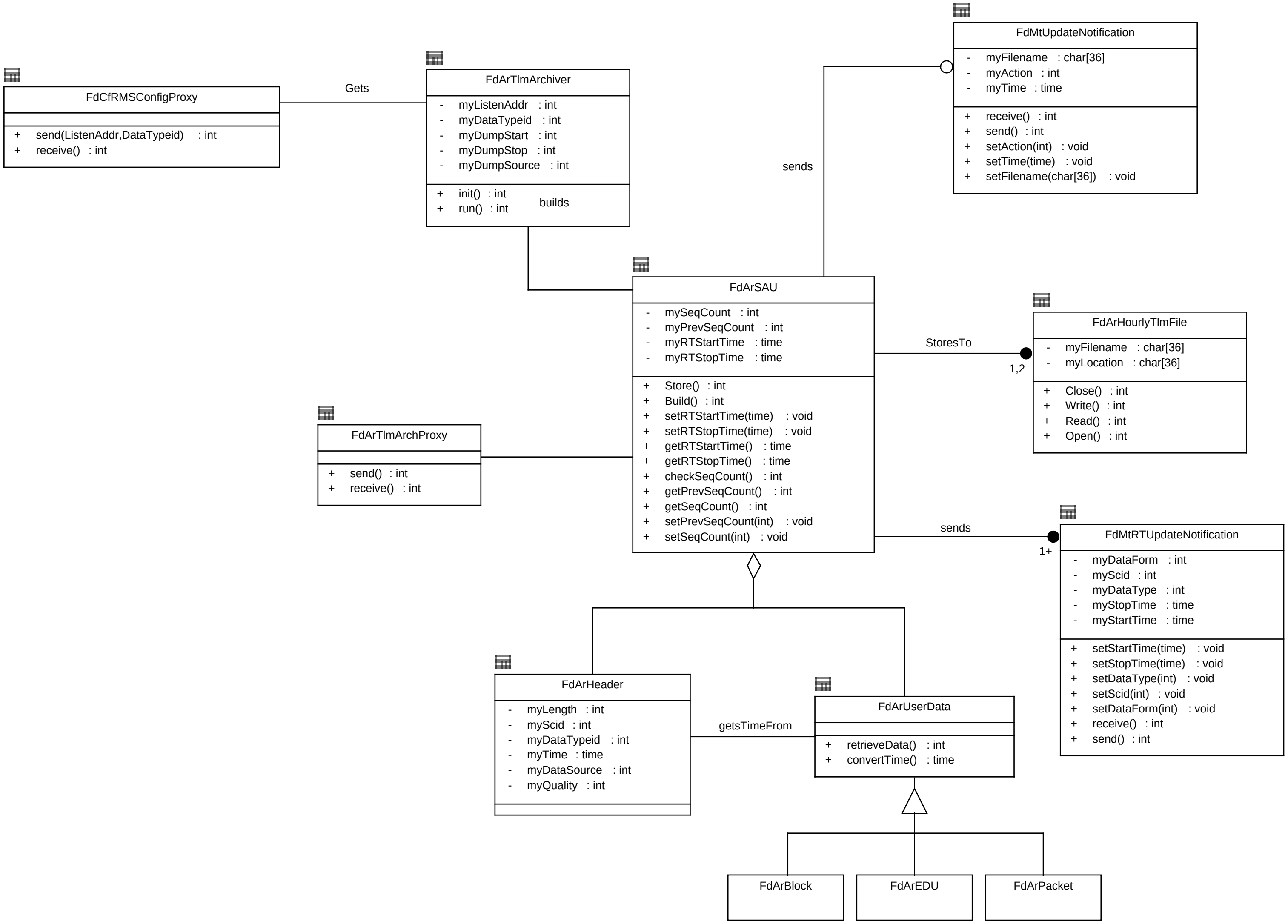


Figure 3.10-2. DMS Telemetry Archiver Object Model

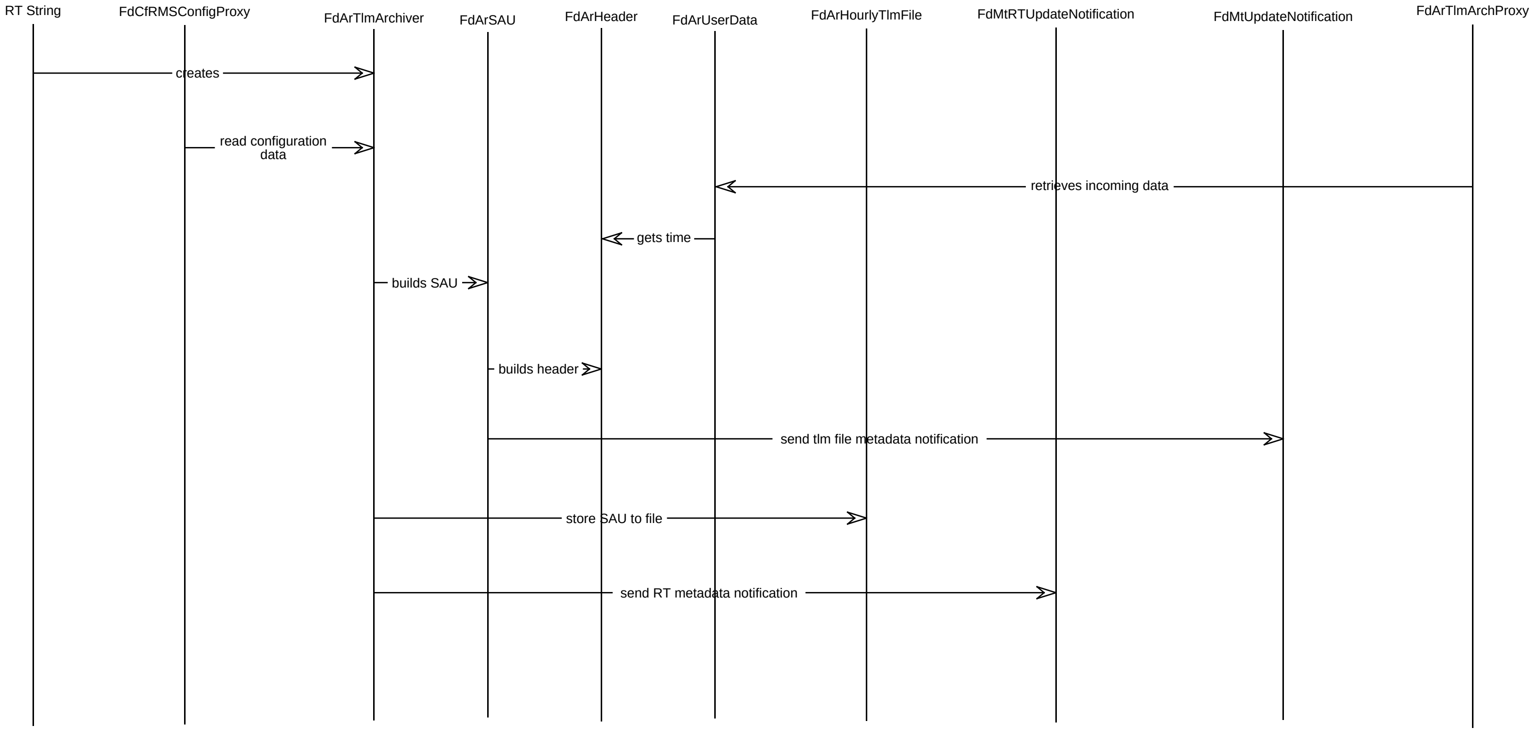


Figure 3.10-3. DMS Telemetry Archiver Event Trace

3.10.4.3 DMS Telemetry Archival Scenario Description

The DMS telemetry archiver is initiated and retrieves its configuration information from RMS via FdCfRMSConfigProxy. The archiver then waits until TLM begins sending data units to the archiver via FdArTlmArchProxy. Upon receipt of data, FdArTlmArchiver calls the build function of FdArSAU to build an SAU from the incoming data unit. A time stamp is placed in the SAU and the SAU header information is built. As each SAU is built, it is archived to an hourly telemetry file via the I/O functions in FdArHourlyTlmFile. File metadata is updated when a new hourly file is opened, and RT/PBK metadata is updated after each real-time contact.

3.10.5 DMS Telemetry Archiver Data Dictionary

FdArHeader

class **FdArHeader**

This class contains the SAU header information

Public Construction

FdArHeader()

This is the default constructor for the class

~FdArHeader()

This is the default destructor for the class

Private Data

int **myDataSource**

This member contains the data source

int **myDataTypeid**

This member contains the data type

int **myLength**

This member contains the data length

int **myQuality**

This member contains the data quality

int **myScid**

This member contains the spacecraft ID

time **myTime**

This member contains the time stamp

FdArHourlyTlmFile

class **FdArHourlyTlmFile**

This class contains the hourly telemetry file

Public Construction

FdArHourlyTlmFile()

This is the default constructor for the class

~FdArHourlyTlmFile()

This is the default destructor for the class

Public Functions

int **Close**(void)

This function closes the hourly tlm file

int **Open**(void)

This function opens the hourly tlm file

int **Read**(void)

This function reads the hourly tlm file

int **Write**(void)

This function writes to the hourly tlm file

Private Data

char **myFilename**[36]

This member variable contains the tlm file name

FdArSAU

class **FdArSAU**

This class contains the Standard Archive Unit for the telemetry archiver, retriever, & playback merger

Public Construction

FdArSAU()

This is the default constructor for the class

~FdArSAU()

This is the default destructor for the class

Public Functions

int **Build**()

This function builds the SAU

int **Store**()

This function stores the SAU

int **checkSeqCount**(int)

This function validates the sequence count

int **getPrevSeqCount**()

This function retrieves the previous seq count

time **getRTStartTime**()

This function retrieves the stream start time

time **getRTStopTime**()

This function retrieves the stream stop time

int **getSeqCount**()

This function retrieves the sequence count

void **setPrevSeqCount**(int)

This function sets the previous sequence counter

void **setRTStartTime**(time)

This function sets the stream start time

void **setRTStopTime**(time)

This function sets the stream stop time

void **setSeqCount**(int)

This function sets the sequence count

Private Data

int **myPrevSeqCount**

This member variable contains the previous sequence count

time **myRTStartTime**

This member variable contains the stream start time

time **myRTStopTime**

This member variable contains the stream stop time

int **mySeqCount**

This member variable contains the current sequence count

FdArTlmArchProxy

class **FdArTlmArchProxy**

Public Construction

FdArTlmArchProxy()

This is the default constructor for the class

~FdArTlmArchProxy()

This is the default destructor for the class

Public Functions

int **receive**()

This receives the configuration information

int **send**(myListenAddr, myDataTypeid)

This sends the configuration information

FdArTlmArchiver

class **FdArTlmArchiver**

This class contains the telemetry archiver

Public Construction

FdArTlmArchiver()

This is the default constructor for the class

~FdArTlmArchiver()

~FdArTlmArchiver();

This is the default destructor for the class

Public Functions

int **init**()

This function initializes the archiver for execution

int **run**()

This function executes the telemetry archiver

Private Data

int **myDataTypeid**

This member contains the data type for this archiver

int **myListenAddr**

This member contains the listen address for this archiver

FdArUserData

class **FdArUserData**

This class contains the EDU data

Public Construction

FdArUserData()

This function is the default constructor

~FdArUserData()

This function is the default destructor

Public Functions

time **convertTime**()

This function converts the time in the EDU to the desired format

int **retrieveData**()

This function retrieves the EDU

FdCfRMSConfigProxy

class **FdCfRMSConfigProxy**

Public Construction

FdCfRMSConfigProxy()

This is the default constructor for the class

~FdCfRMSConfigProxy()

This is the default destructor for the class

Public Functions

int **receive**()

This receives the configuration information

int **send**(myListenAddr, myDataTypeid)

This sends the configuration information

FdMtRTUpdateNotification

class **FdMtRTUpdateNotification**

This class contains the interface between the playback merger and the RT/PBK metadata

Public Construction

FdMtRTUpdateNotification()

This is the default constructor for the class

~FdMtRTUpdateNotification()

This is the default destructor for the class

Public Functions

int **receive**()

This function receives the notification

int **send**()

This function sends the notification

void **setDataForm**(int)

This function sets the Data Form attribute

void **setDataType**(int)

This function sets the Data Type attribute

void **setScid**(int)

This function sets the SCID attribute

void **setStartTime**(time)

This function sets the Start Time attribute

void **setStopTime**(time)

This function sets the Stop Time attribute

Private Data

int **myDataForm**

This member variable contains the stream form (RT vs PBK)

int **myDataType**

This member variable contains the stream data type

int **myScid**

This member variable contains the stream SC id

time **myStartTime**

This member variable contains the stream start time

time **myStopTime**

This member variable contains the stream stop time

FdMtUpdateNotification

class **FdMtUpdateNotification**

This class contains the interface between the playback merger and the DMS metadata

Public Construction

FdMtUpdateNotification()

This is the default constructor for the class

~FdMtUpdateNotification()

This is the default constructor for the class

Public Functions

int **receive**(void)

This function receives the notification

int **send**(void)

This function sends the notification

void **setAction**(int)

This function sets the Action attribute

void **setFilename**(char)

This function sets the Filename attribute

void **setTime**(time)

This function sets the Time attribute

Private Data

int **myAction**

This member contains the requested action

char **myFilename**[36]

This member contains the filename for metadata

time **myTime**

This member contains the Time

3.11 DMS Telemetry Playback Merger

The DMS Telemetry Playback Merger is a persistent process responsible for receiving telemetry housekeeping playback files from EDOS and then merging them with the existing hourly telemetry files. The playback EDUs are stored in the archive and the telemetry from previous real-time contacts is archived only if the playback file has a sequence gap or bad quality. As each hourly telemetry file is filled, it is then saved in both local and long-term storage.

3.11.1 DMS Playback Merger Context

The DMS Playback Merger interfaces are described below and displayed in the context diagram.

EDOS:

Sends a notification to the playback merger once a playback file has been sent to us and is available to be merged.

Long-term Storage:

Receives notification from the playback merger that a complete hourly telemetry file is available to be copied to long-term storage.

Analysis:

Receives notification from the playback merger that an hourly telemetry file has been successfully merged and is ready for Analysis to perform statistics on.

3.11.2 DMS Telemetry Playback Merge Interfaces

Table 3.11-1. Telemetry Playback Merge Interfaces

Interface Service	Interface Class	Interface Class Description	Service Provider	Service User	Frequency
Send hourly files to SCDO	FdLTScdoSend	Notify SCDO of new hourly file.	DMS	DMS	Approx. 2/ day
Inform Analysis that a tlm file is ready for statistics.	FoArAnaIF	Notify Analysis subsystem of new hourly tlm file.	ANA	DMS	Approx. 2/ day
Inform DMS that a playback file is ready to be merged.	FdArEDOSPbkIF	Notify DMS that a playback file is ready.	EDOS	DMS	Approx. 2/ day

Note: Above table is subject to change.

3.11.3 DMS Telemetry Playback Merge Object Model

The FdArPbkMerger object merges playback housekeeping data with existing real-time and playback data in a seamless archive. FdArPbkMerger uses the receive function of FdArEDOSPbkIF to await notification from EDOS when a playback file is available to be merged. Two FdArHourlyTlmFile objects are utilized - one for the existing hourly file and one for the temporary file which will contain the merged data and which will eventually supersede the existing hourly file. The FdArPbkFile object is utilized to access the playback files and read the EDUs from them. Once the EDUs are read, FdArPbkMerger uses FdArSAU.build to build the SAUs from the EDUs. If a sequence gap exists in the playback file, or if the playback SAU is of bad quality, then the existing hourly telemetry file is opened and the matching SAU is retrieved (if it exists). In cases where a sequence gap occurs which cannot currently be filled, FdArPbkMerger will move on and call FdMtRTUpdateNotification.send to update the telemetry metadata to indicate that a gap exists. The FdArUserData.convertTime function is used to verify that the spacecraft time of the EDU is valid. When the top of an hour is reached, FdArPbkMerger calls FdMtUpdateNotification to update metadata to reference the completed hourly file. Also, FdArPbkMerger calls FoArAnaIF.send to notify Analysis that a new hourly telemetry file is now complete and ready for statistics calculations. When an hourly telemetry file is completely filled, FdArPbkMerger also calls FdLTScdoSend.send to notify long-term storage that a complete telemetry file is ready to be copied over to the long-term archive.

The FdArSAU object is used to build and store the SAUs. It also maintains the current telemetry sequence count and verifies that the EDU sequence count is in order. In addition, FdArSAU verifies, updates and maintains the start & stop time of the current, contiguous, telemetry stream.

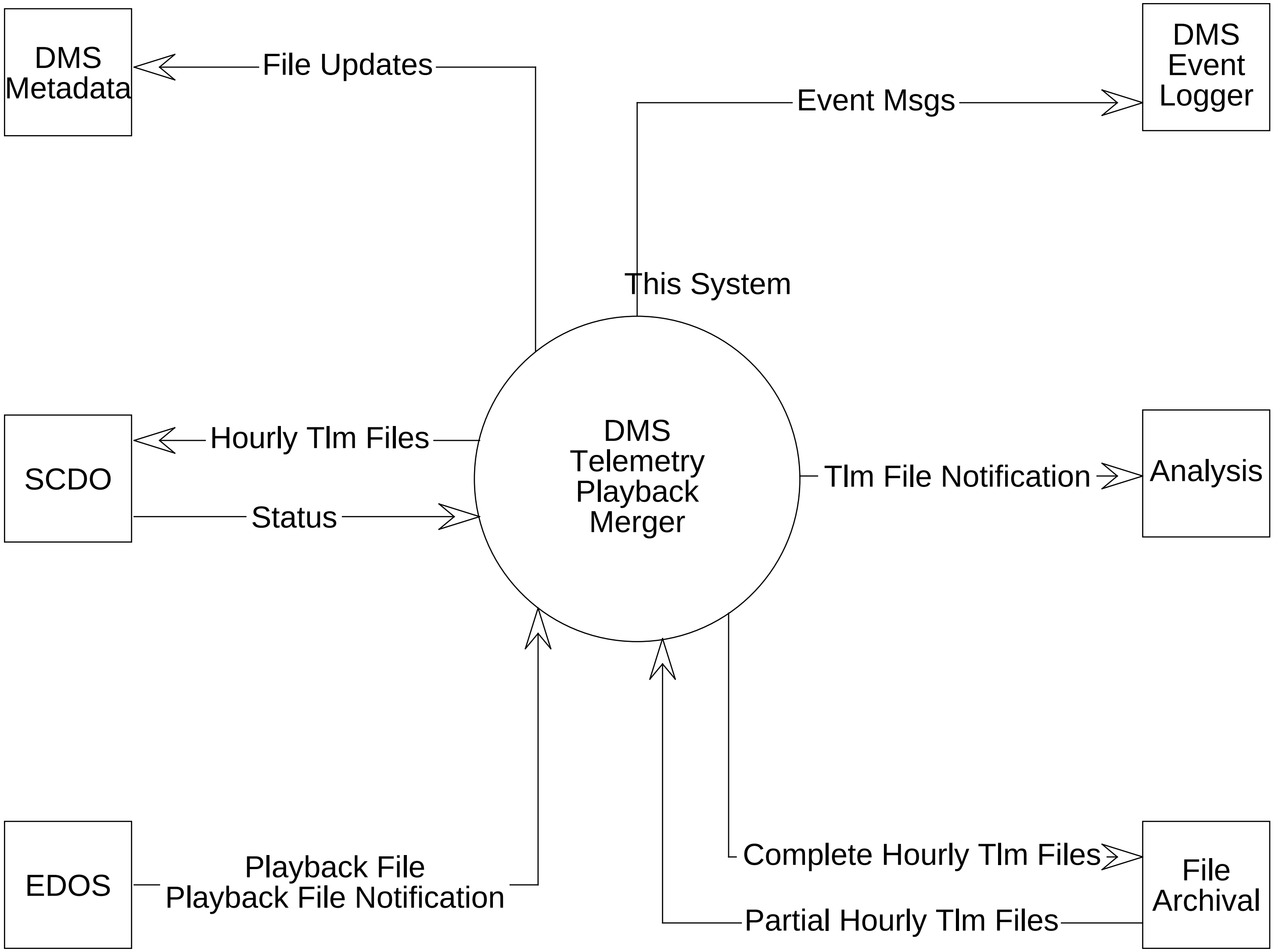


Figure 3.11-1. DMS Telemetry Playback Merger Context Diagram

The FdArHourlyTlmFile object is responsible for operations on the archive file. The FdMtUpdateNotification object provides the interface between the archiver and the archive metadata table. This notification is sent whenever a new file is opened. The FdMtRTUpdateNotification object provides the interface to the 'available telemetry' metadata table. This table contains start and stop times of all contiguous telemetry data in the archive. The FdArPbkFile object is used to perform I/O operations on the playback file received from EDOS.

The FdArEDOSPbkIF object is used to notify the playback merger that a playback file exists and is ready to be merged. The FdLTScdoSend object is utilized to notify long-term storage that a completed hourly telemetry file is ready to be copied over to long-term storage.

3.11.4 DMS Telemetry Playback Merger Dynamic Model

3.11.4.1 Telemetry Playback Merger Scenario 1

3.11.4.1.1 Telemetry Playback Merger Scenario 1 Abstract

The Playback Merger scenario 1 describes the receipt of a complete (i.e. no sequence gaps) playback file from EDOS, the reading of this file and its merge into the existing archive. The scenario also describes the merge activities of notifying the interested parties of the newly-created, complete hourly telemetry files.

3.11.4.1.2 Telemetry Playback Merger Scenario 1 Summary Information

Interfaces:

EDOS

Analysis

SCDO

Stimulus:

Receipt of a playback file from EDOS.

Desired Response:

Seamless, merged archive of playback data with existing real-time and playback data.

Notification to Analysis that new hourly telemetry files are ready for statistics.

Notification to SCDO that new hourly telemetry files are ready for long-term storage.

Update metadata to accurately reflect all real-time and playback data currently in the system and available for use.

Pre-Conditions:

Playback merger software has been initiated

Post-conditions:

3.11.4.1.3 Telemetry Playback Merger Scenario 1 Description

The playback merger receives notification from EDOS that a playback file has been sent over and is ready for processing. The playback merger then instantiates two FdArHourlyTlmFile objects. The first instance is for a temporary hourly file which will eventually contain all of the merged data. The second instance is for the hourly file whose time corresponds with the start time of the playback file data. The existing hourly file is opened and its playback contents are first copied into the temporary hourly file.

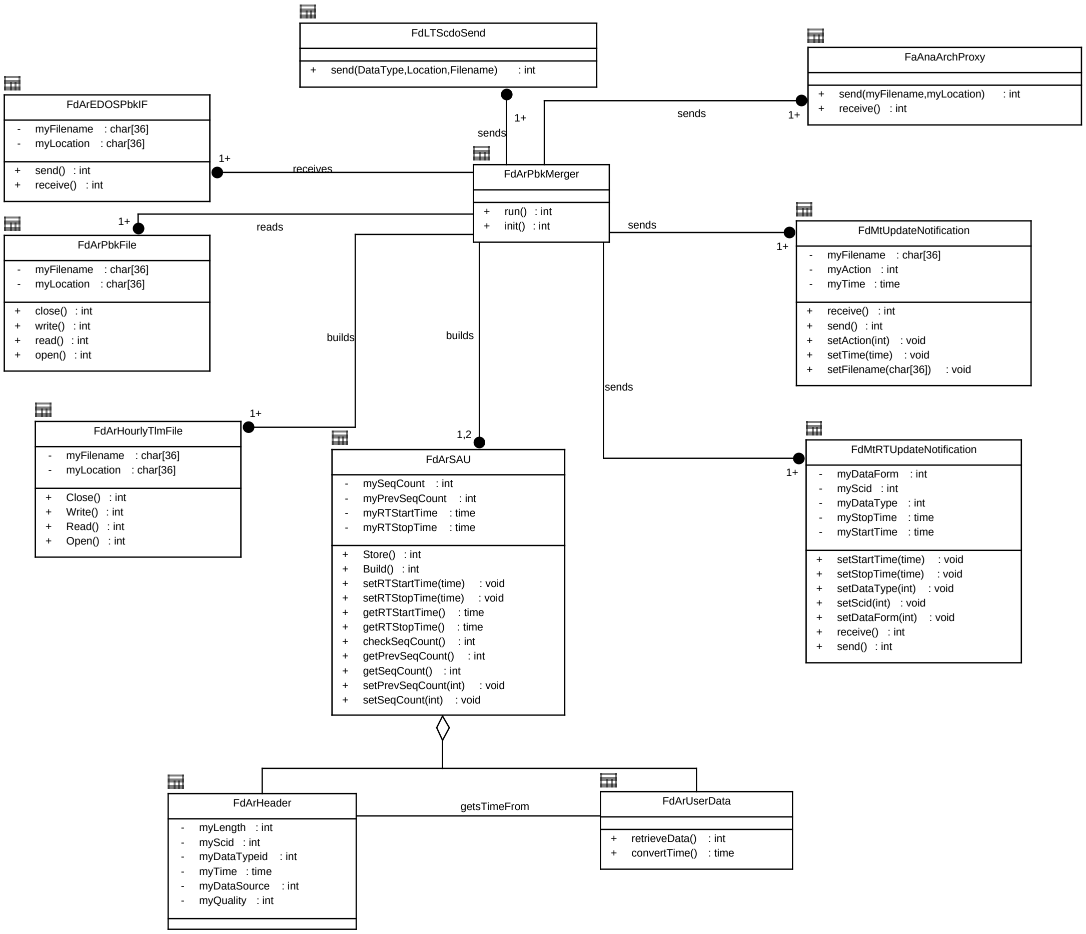


Figure 3.11-2. DMS Telemetry Playback Merger Object Model

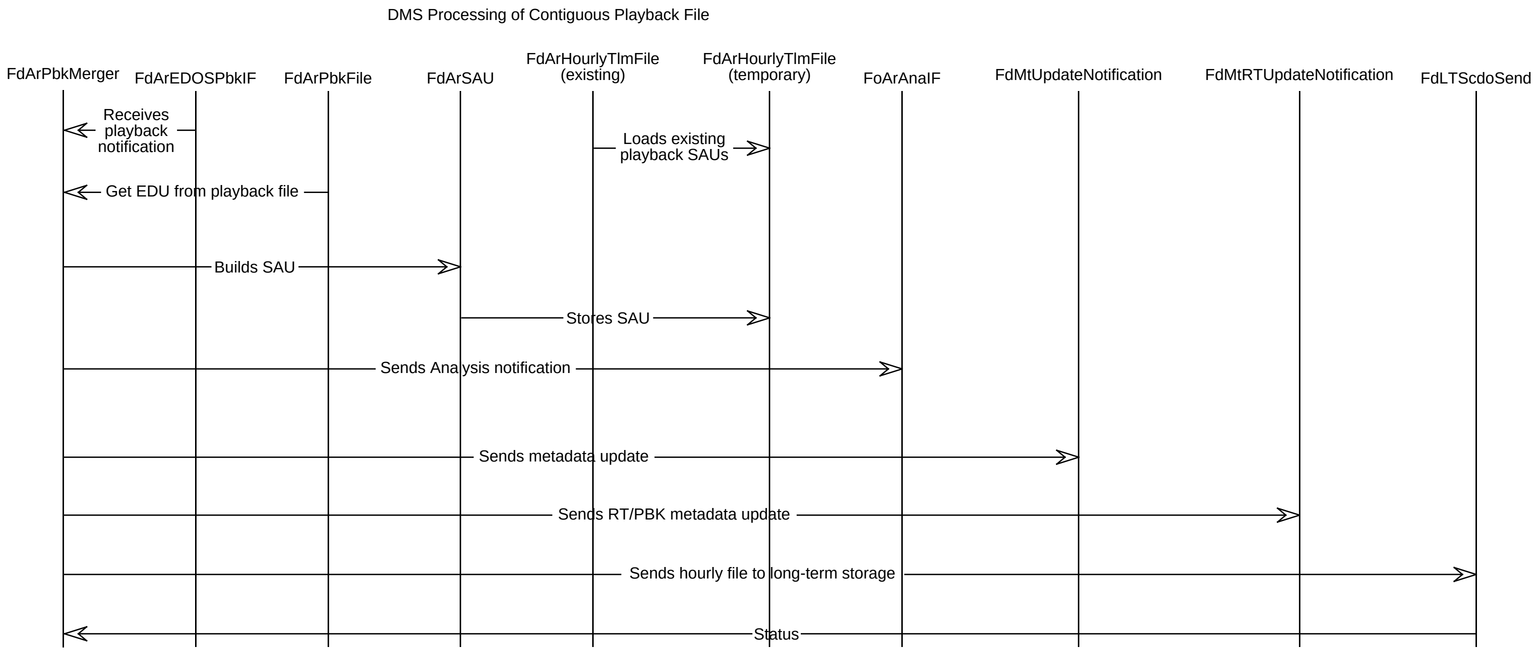

DMS Processing of Contiguous Playback File
FdArPbkMerger
FdArEDOSPbkIF
FdArPbkFile
FdArSAU
FdArHourlyTlmFile (existing)
FdArHourlyTlmFile (temporary)
FoArAnaIF
FdMtUpdateNotification
FdMtRTUpdateNotification
FdLTScdoSend
Receives playback notification
Loads existing playback SAUs
Get EDU from playback file
Builds SAU
Stores SAU
Sends Analysis notification
Sends metadata update
Sends RT/PBK metadata update
Sends hourly file to long-term storage
Status

Figure 3.11-3. DMS Telemetry Playback Merger Scenario 1 Event Trace

The playback merger instantiates an FdArPbkFile object to access the playback file. The EDUs are read from the playback file and the playback merger instantiates an FdArSAU object to build the playback Standard Archive Units. The sequence counters are checked, as is the spacecraft time, to ensure that the playback data has no gaps and that the times are correct. As each hour of data is merged, the temporary telemetry file supersedes the existing hourly file. This processing is repeated for each EDU in the playback file.

As each hour is completed, the playback merger uses an instance of FoArAnaIF to notify Analysis that an existing hourly file is ready for statistics generation. Telemetry file metadata is also updated using an instance of FdMtUpdateNotification and FdMtRTUpdateNotification (for real-time/ playback availability). Finally, the playback merger notifies SCDO of the hourly file via an instance of the FdLTScdoSend class.

3.11.4.2 Telemetry Playback Merger Scenario 2

3.11.4.2.1 Telemetry Playback Merger Scenario 2 Abstract

The Playback Merger scenario 2 describes the receipt of a playback file from EDOS which contains a sequence gap, the reading of this file and its merge into the existing archive. The scenario also describes the merge activities of notifying the interested parties of the newly-created, complete hourly telemetry files.

3.11.4.2.2 Telemetry Playback Merger Scenario 2 Summary Information

Interfaces:

EDOS

Analysis

SCDO

Stimulus:

Receipt of a playback file from EDOS.

Desired Response:

Seamless, merged archive of playback data with existing real-time and playback data.

Notification to Analysis that new hourly telemetry files are ready for statistics.

Notification to SCDO that new hourly telemetry files are ready for long-term storage.

Update metadata to accurately reflect all real-time and playback data currently in the system and available for use.

Pre-Conditions:

Playback merger software has been initiated

Post-conditions:

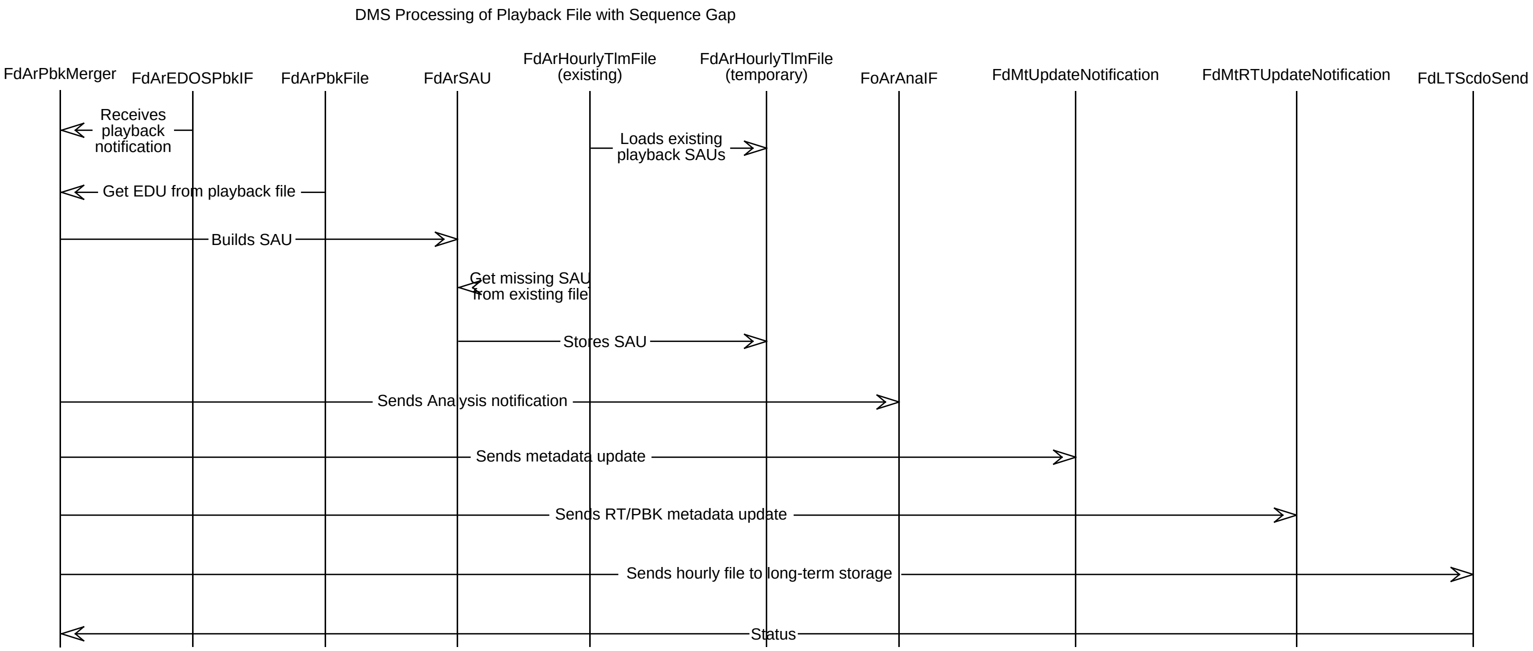

DMS Processing of Playback File with Sequence Gap
FdArPbkMerger
FdArEDOSPbkIF
FdArPbkFile
FdArSAU
FdArHourlyTlmFile (existing)
FdArHourlyTlmFile (temporary)
FoArAnaIF
FdMtUpdateNotification
FdMtRTUpdateNotification
FdLTScdoSend
Receives playback notification
Loads existing playback SAUs
Get EDU from playback file
Builds SAU
Get missing SAU from existing file
Stores SAU
Sends Analysis notification
Sends metadata update
Sends RT/PBK metadata update
Sends hourly file to long-term storage
Status

Figure 3.11-4. DMS Telemetry Playback Merger Scenario 2 Event Trace

3.11.4.2.3 Telemetry Playback Merger Scenario 2 Description

The playback merger receives notification from EDOS that a playback file has been sent over and is ready for processing. The playback merger instantiates two FdArHourlyTlmFile objects. The first instance is for a temporary hourly file which will eventually contain all of the merged data. The second instance is for the hourly file whose time corresponds with the start time of the playback file data. The existing hourly file is opened and its playback contents are first copied into the temporary hourly file.

The playback merger instantiates an FdArPbkFile object to access the playback file. The EDUs are read from the file and the playback merger instantiates an FdArSAU object to build the playback Standard Archive Units. The sequence counter check indicates a sequence gap (the playback file is missing one or more EDUs). At this point, the appropriate, existing hourly file is opened and is searched for the missing EDU(s). In this scenario, the missing EDU(s) are located and stored to the temporary file from the existing hourly file. When the gap has been filled, processing continues as before.

As each hour is completed, the playback merger uses an instance of FoArAnaIF to notify Analysis that an existing hourly file is ready for statistics generation. Telemetry file metadata is also updated using an instance of FdMtUpdateNotification and FdMtRTUpdateNotification (for real-time/ playback availability). Finally, the playback merger notifies SCDO of the hourly file via an instance of the FdLTScdoSend class.

3.11.5 DMS Telemetry Playback Merger Data Dictionary

FdArEDOSPbkIF

class **FdArEDOSPbkIF**

This class contains the interface between EDOS & the playback merger task.

Public Construction

FdArEDOSPbkIF()

This function is the default constructor for the class

~FdArEDOSPbkIF()

This function is the default destructor for the class

Public Functions

int **receive**()

This function receives notification from EDOS

int **send**()

This function sends notification to the merger.

Private Data

char **myFilename**[36]

This member variable contains the playback file name

char **myLocation**[36]

This member variable contains the playback file location

FdArHeader

class **FdArHeader**

This class contains the SAU header information

Public Construction

FdArHeader()

This is the default constructor for the class

~FdArHeader()

This is the default destructor for the class

Private Data

int **myDataSource**

This member contains the data source

int **myDataTypeid**

This member contains the data type

int **myLength**

This member contains the data length

int **myQuality**

This member contains the data quality

int **myScid**

This member contains the spacecraft ID

time **myTime**

This member contains the time stamp

FdArHourlyTlmFile

class **FdArHourlyTlmFile**

This class contains the hourly telemetry file

Public Construction

FdArHourlyTlmFile()

This is the default constructor for the class

~FdArHourlyTlmFile()

This is the default destructor for the class

Public Functions

int **Close**(void)

This function closes the hourly tlm file

int **Open**(void)

This function opens the hourly tlm file

int **Read**(void)

This function reads the hourly tlm file

int **Write**(void)

This function writes to the hourly tlm file

Private Data

char **myFilename**[36]

This member variable contains the tlm file name

FdArPbkFile

class **FdArPbkFile**

This class contains the playback file

Public Construction

FdArPbkFile()

This function is the default constructor for the class

~FdArPbkFile()

FdArPbkFile

This function is the default destructor for the class

Public Functions

int **close**()

This function closes the playback file

int **open**()

This function opens the playback file

int **read**()

This function reads the playback file

int **write**()

This function writes to the playback file

Private Data

char **myFilename**[36]

This member variable contains the playback file name

char **myLocation**[36]

This member variable contains the playback file location

FdArPbkMerger

class **FdArPbkMerger**

This class represents the playback merger task

Public Construction

FdArPbkMerger()

This is the default constructor for the class

~FdArPbkMerger()

This is the default destructor for the class

Public Functions

int **init**()

This function initializes the playback merger

int **run**()

This function runs the playback merger

FdArSAU

class **FdArSAU**

This class contains the Standard Archive Unit for the telemetry archiver, retriever, & playback merger

Public Construction

FdArSAU(listenAddr)

This is the default constructor for the class

~FdArSAU()

This is the default destructor for the class

Public Functions

int **Build**(void)

This function builds the SAU

int **Store**(void)

This function stores the SAU

int **checkSeqCount**()

This function validates the sequence count

int **getPrevSeqCount**()

This function retrieves the previous seq count

time **getRTStartTime**()

This function retrieves the stream start time

time **getRTStopTime**()

This function retrieves the stream stop time

int **getSeqCount**()

This function retrieves the sequence count

void **setPrevSeqCount**(int)

This function sets the previous sequence counter

void **setRTStartTime**(time)

This function sets the stream start time

void **setRTStopTime**(time)

This function sets the stream stop time

void **setSeqCount**(int)

This function sets the sequence count

Private Data

int **myPrevSeqCount**

This member variable contains the previous sequence count

time **myRTStartTime**

This member variable contains the stream start time

time **myRTStopTime**

This member variable contains the stream stop time

int **mySeqCount**

This member variable contains the current sequence count

FdArUserData

class **FdArUserData**

This class contains the EDU data

Public Construction

FdArUserData()

This function is the default constructor

~FdArUserData()

This function is the default destructor

Public Functions

time **convertTime**()

This function converts the time in the EDU to the desired format

int **retrieveData**()

This function retrieves the EDU

FdLTScdoSend

class **FdLTScdoSend**

This class represents the interface between the playback merger and SCDO

Public Construction

FdLTScdoSend()

This is the default constructor for the class

~FdLTScdoSend()

This is the default destructor for the class

Public Functions

int **send**(void)

This class sends notification to SCDO

Private Data

int **myDataType**

This member contains the data type of the file

char **myFilename**[36]

This member contains the name of the file

char **myLocation**[36]

This member contains the location of the file

FdMtRTUpdateNotification

class **FdMtRTUpdateNotification**

This class contains the interface between the playback merger and the RT/PBK metadata

Public Construction

FdMtRTUpdateNotification()

This is the default constructor for the class

~FdMtRTUpdateNotification()

This is the default destructor for the class

Public Functions

int **receive**()

This function receives the notification

int **send**()

This function sends the notification

void **setDataForm**(int)

This function sets the Dataform attribute

void **setDataType**(int)

This function sets the DataType attribute

void **setScid**(int)

This function sets the Scid attribute

void **setStartTime**(time)

This function sets the StartTime attribute

void **setStopTime**(time)

This function sets the StopTime attribute

Private Data

int **myDataForm**

This member variable contains the stream form (RT vs PBK)

int **myDataType**

This member variable contains the stream data type

int **myScid**

This member variable contains the stream SC id

time **myStartTime**

This member variable contains the stream start time

time **myStopTime**

This member variable contains the stream stop time

FdMtUpdateNotification

class **FdMtUpdateNotification**

This class contains the interface between the playback merger and the DMS metadata

Public Construction

FdMtUpdateNotification()

This is the default constructor for the class

~FdMtUpdateNotification()

This is the default constructor for the class

Public Functions

int **receive**()

This function receives the notification

int **send**()

This function sends the notification

void **setAction**(int)

This function sets the Action attribute

void **setFilename**(char)

This function sets the Filename attribute

void **setTime**(time)

This function sets the Time attribute

Private Data

int **myAction**

This member contains the requested action

char **myFilename**[36]

This member contains the filename for metadata

time **myTime**

This member contains the Time

FoArAnaIF

class **FoArAnaIF**

This class contains the interface between Analysis and the playback merger

Public Construction

FoArAnaIF()

This is the default constructor for the class

~FoArAnaIF()

FoArAnaIF

This is the default destructor for the class

Public Functions

int **receive**()

This function receives the notification

int **send**()

This function sends the notification to Analysis

Private Data

char **myFilename**[36]

This member variable contains the tlm file name

char **myLocation**[36]

This member variable contains the tlm file location

3.12 DMS Telemetry Retrieval

The DMS Telemetry Retrieval process is a persistent process responsible for accepting user replay requests (Shared, Dedicated, or Analysis requests) and serving the data to the appropriate Analysis and Telemetry processes required to process the replay data. The user specifies the various parameters surrounding the replay request and the telemetry retrieval process ensures that the requested data is retrieved (either locally or from long-term storage) and served. The telemetry retrieval process places the user requests in a queue and maintains this queue over the life of the process while providing users with the ability to look into the queue and view the status of their respective requests.

3.12.1 DMS Telemetry Retrieval Context

The DMS Telemetry Retrieval interfaces are described below and displayed in the context diagram.

Long-term Storage:

Receives a request from the telemetry retrieval process that one or more requested telemetry files need to be transferred from long-term to local storage.

Sends files and notification once all requested files have been sent over.

Analysis:

The Analysis request manager receives the analysis request from the telemetry retrieval request queue manager. The request manager sends replay status back to the request queue manager.

The Analysis cruncher sends a request for EDUs to the Data Retriever

The Analysis cruncher receives EDUs from the Data Retriever.

Telemetry:

Receives telemetry EDUs.

FUI:

Sends user replay requests to the request queue manager.

Sends requests for the status of previously submitted replay requests.

Receives status on previously submitted replay requests.

Sends start/pause/step requests to the Data Retriever(s).

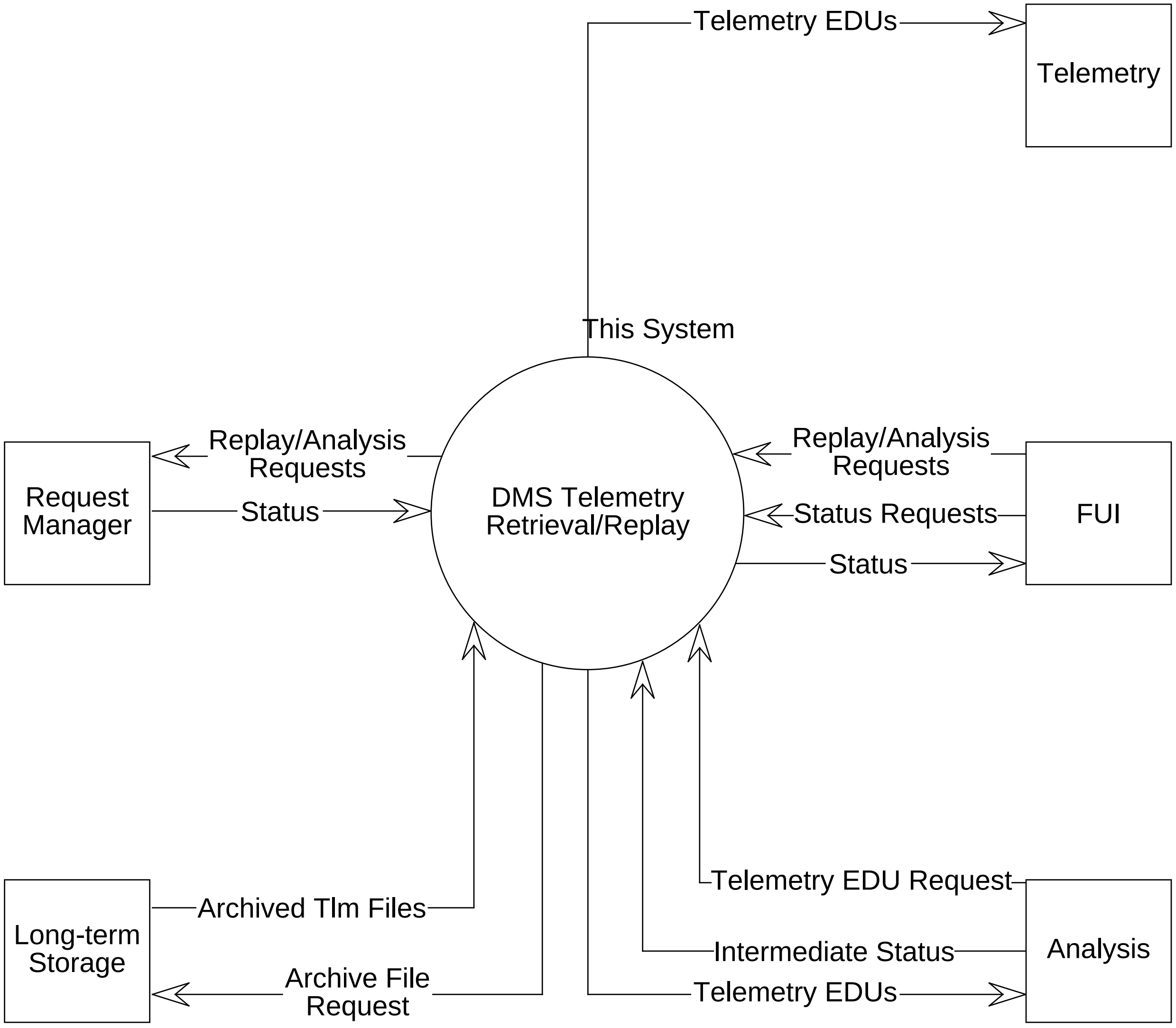


Figure 3.12-1. DMS Telmetry Retrieval Context Diagram

3.12.2 DMS Telemetry Retrieval Interfaces

Table 3.12-1. DMS Telemetry Retrieval Interfaces

Interface Service	Interface Class	Interface Class Description	Service Provider	Service User	Frequency
Send request info to DMS	FaRpFUIToQueueProxy	Provide interface between FUI & DMS	DMS	FUI	Frequently
	FdRqReplayRequest	Specific request information			
Send request status info to FUI	FaRpQueueToFUIProxy	Provide interface between DMS & FUI	FUI	DMS	Frequently
	FoDsReplayStatus	Status of the replay request			
	FoDsReplayString	String information of the replay request			
Send request status to FUI	FaRpAnalysisStatus	Provide analysis request status info to FUI	FUI	DMS	Frequently
Send analysis request to the request manager	FaRpQueueToReqMgrProxy	Provide interface between DMS and the request manager for analysis requests	ANA	DMS	Frequently
	FdRqReplayRequest	Specific request information			
Send messages and status from the request mgr to DMS	FaRpReqMgrToQueueProxy	Provide interface between request manager & DMS	DMS	ANA	Frequently
	FoDsReplayStatus	Status of the replay request			
	FoDsReplayString	String information of the replay request			
Allow FUI to control replay data flow	FdRqFUIToDataRetrieverProxy	Provide interface between FUI and DMS data retriever	DMS	FUI	Frequently
Allow Analysis to control analysis request data flow	FdRpReqMgrToDRProxy	Provide interface between analysis cruncher and DMS data retriever	DMS	FUI	Frequently

Note: Above table is subject to change.

3.12.3 DMS Telemetry Retrieval Object Model

The FoDsRequestQueueMgr object queues up, submits, and monitors all user replay requests (Shared, Dedicated, Analysis). FoDsRequestQueueMgr uses the FaRpFUIToQueueProxy to receive incoming requests (contained in the FdRqReplayRequest object). The FoDsRequestQueueMgr.dbLookup function is called to partition the request by its separate database components (if the request crosses over a database boundary). For analysis requests, the findStation function is then invoked in order to determine upon which machine to execute the request. Once this information is determined then the request queue, FoDsRequestQueue, is updated to include this new request. FoDsRequestQueueMgr also checks to see if the requested telemetry files are located at long-term storage. If so, then FdLTScdoRetrieve is used to retrieve those files from SCDO.

For analysis requests and dedicated replay requests, the request information is then passed via FaRpQueueToReqMgrProxy to the FaAnRequestMgr on the selected machine. FaAnRequestMgr will then initiate FoDsDataRetriever on the user station to serve the telemetry data. For shared replay requests, FoDsRequestQueueMgr initiates the FoDsDataRetriever process on the Data Server.

The initialization status of analysis and dedicated replays requests is contained in FoDsReplayStatus and is sent back from FaAnRequestMgr to FoDsRequestQueueMgr via FaRpReqMgrToQueueProxy. FoDsRequestQueueMgr determines the initialization status for shared replay requests. FoDsRequestQueueMgr sends the status back to FUI via FaRpQueueToFUIProxy. If the startup was successful, then FUI initiates the replay using FoRqFUIToDataRetrieverProxy. This proxy allows FUI to start and stop the data flow from the Data Retriever. Once FoDsDataRetriever receives the start request from FUI, it begins reading the telemetry files using FdArHourlyTlmFile functions and building EDUs from the SAUs using FtDsEDU.build. The Data Retriever then sends the EDUs to the appropriate destination address by invoking the sendTlmEDU member function.

FUI can halt and restart the replay process using the FoRqFUIToDataRetrieverProxy, or it can cancel the replay via the FaRpFUIToQueueProxy. When the replay is canceled, then FoDsRequestQueueMgr forwards on the cancellation request to FaAnReqeuestMgr and deletes the request from the queue by calling the deleteFromQueue function of FoDsRequestQueueMgr.

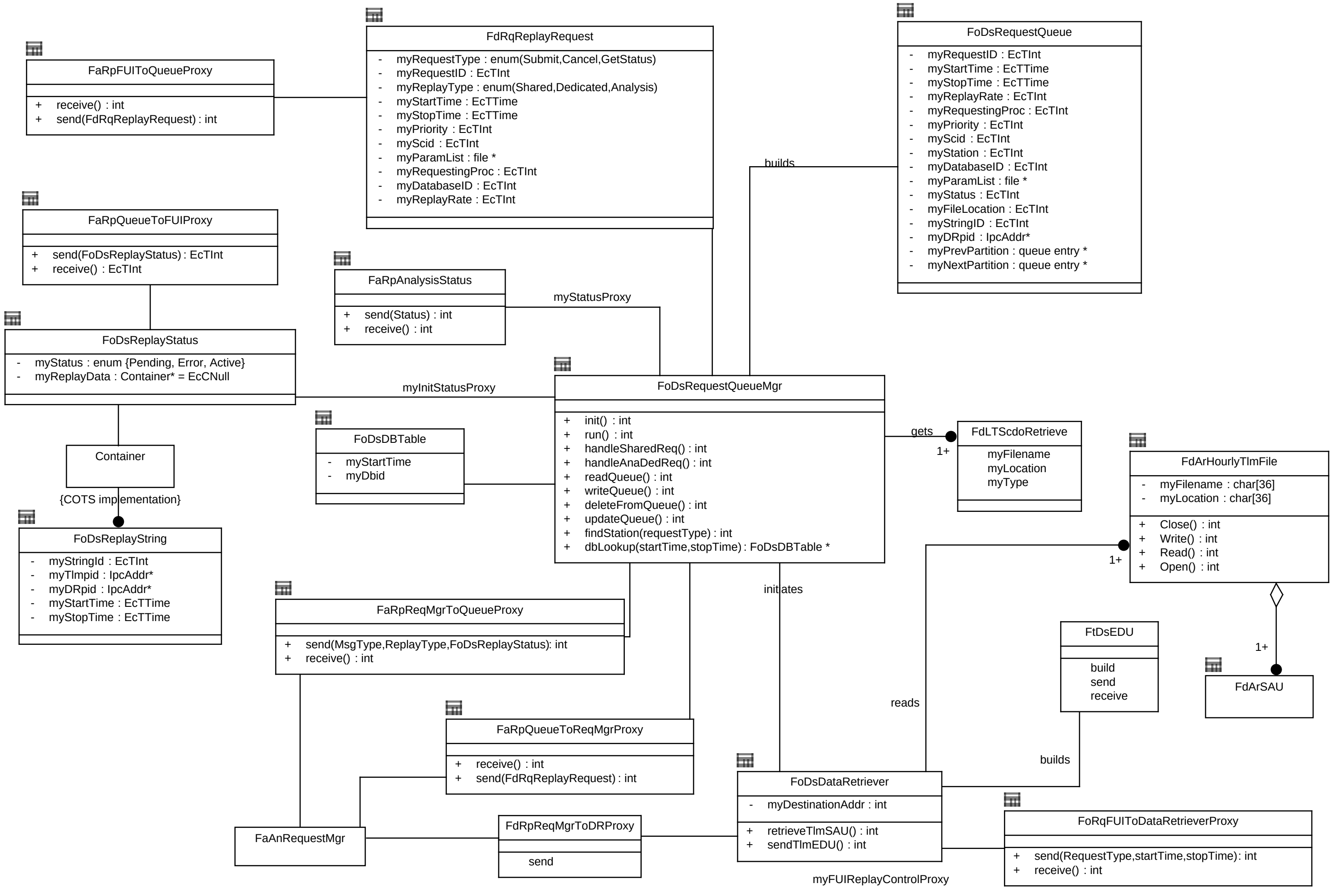


Figure 3.12-2. DMS Telemetry Retrieval Object Model

3.12.4 DMS Telemetry Retrieval Dynamic Model

3.12.4.1.1 Telemetry Retrieval/Replay Scenario 1 Abstract

The Telemetry Retrieval/Replay scenario 1 describes the processing of an Analysis request from FUI.

3.12.4.1.2 Telemetry Retrieval/Replay Scenario 1 Summary Information

Interfaces:

FUI

Analysis

SCDO

RMS

Stimulus:

Receipt of an Analysis request from FUI.

Desired Response:

Initialize logical string components, enter request onto the queue, & serve the requested data to the specified analysis process(es).

Upon completion of request, delete entry from the queue.

Pre-Conditions:

Request Queue manager software has been initiated.

Post-conditions:

3.12.4.1.3 Telemetry Retrieval Scenario 1 Description

The Request Queue Manager receives an analysis request from FUI. The Request Queue Manager adds the request to the request queue. A quick look is taken to determine the location of the necessary telemetry files. If one or more are located at long-term storage, then FdLTScdoRetrieve is used to request the needed files. A status is returned from SCDO once the transfer is complete. The request is then submitted to the Request Manager on an available user station. The Request Manager creates an FoDsDataRetriever process (as well as an Analysis & Telemetry process). The status of the initialization is then determined by the Request Manager. This status is sent to the Request Queue Manager, who updates the appropriate entry in the request queue and returns the status information to FUI.

If the initialization was successful, then FUI sends a start request to the appropriate FoDsDataRetriever process. The Data Retriever then begins reading the telemetry files and serving the EDUs to the Analysis cruncher process. When the replay is complete, the Request Manager sends the completion status back to the Request Queue Manager. The Request Queue Manager passes the completion status to FUI and then deletes the entry from the request queue.

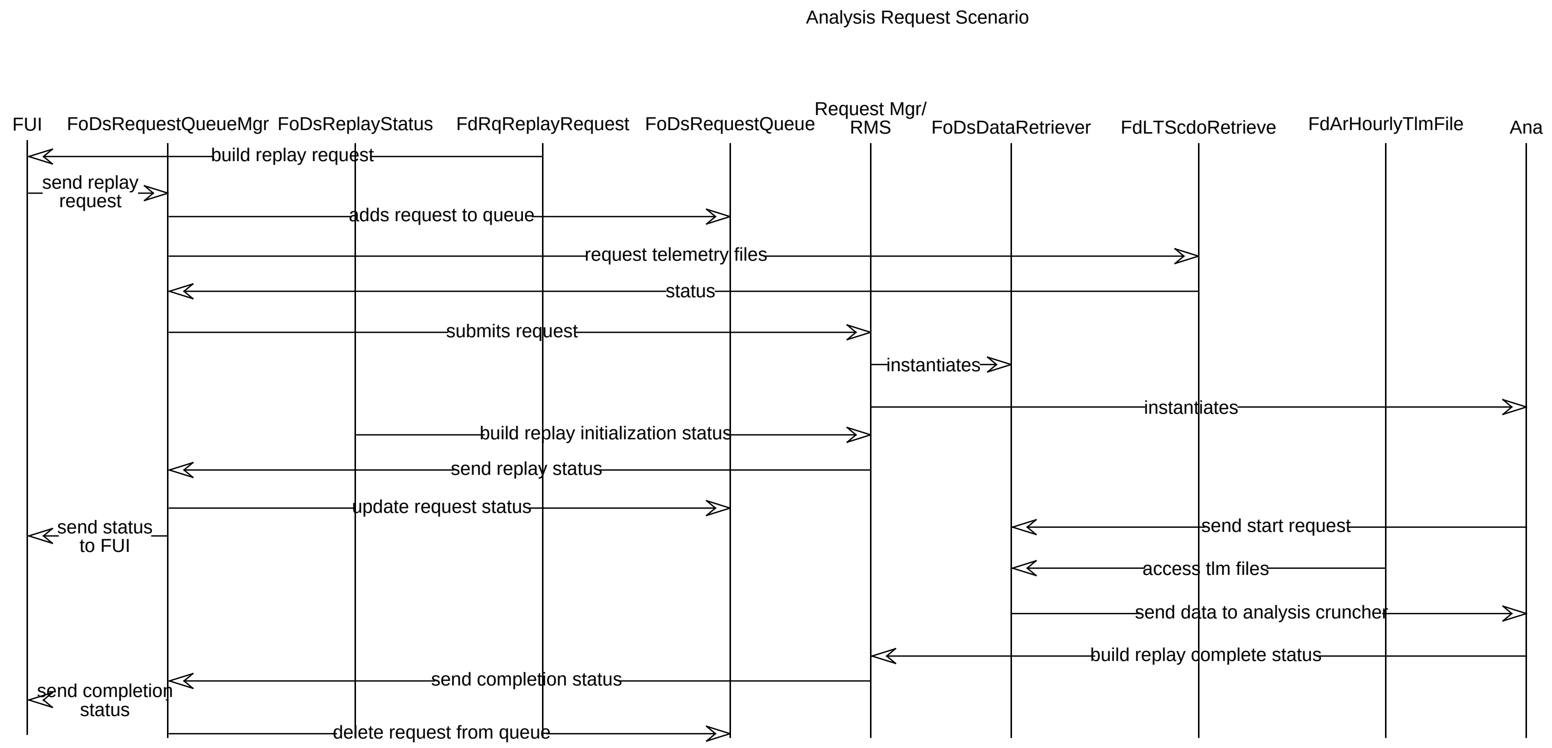


Figure 3.12-3. DMS Telemetry Retrieval Scenario 1 Event Trace

3.12.4.2.1 Telemetry Retrieval/Replay Scenario 2 Abstract

The Telemetry Retrieval/Replay scenario 2 describes the processing of a Dedicated replay request from FUI.

3.12.4.2.2 Telemetry Retrieval/Replay Scenario 2 Summary Information

Interfaces:

FUI

Analysis

TLM

SCDO

RMS

Stimulus:

Receipt of a Dedicated replay request from FUI.

Desired Response:

Initialize logical string components, enter request onto the queue, & serve the requested data to the specified telemetry process.

Upon completion of request, delete entry from the queue.

Pre-Conditions:

Request Queue manager software has been initiated.

Post-conditions:

3.12.4.2.3 Telemetry Retrieval/Replay Scenario 2 Description

The request queue manager receives a dedicated replay request from FUI. The Request Queue Manager adds the request to the request queue. A quick look is taken to determine the location of the necessary telemetry files. If one or more are located at long-term storage, then FdLTScdoRetrieve is used to request the needed files. A status is returned from SCDO once the transfer is complete. The request is then submitted to the Request Manager on an available user station. The Request Manager creates an FoDsDataRetriever process and a Telemetry process. The status of the initialization is then determined by the Request Manager. This status is sent to FoDsRequestQueueMgr, who updates the appropriate entry in the request queue and returns the request status information to FUI.

If the initialization was successful, then FUI sends a start request to the appropriate FoDsDataRetriever process. The Data Retriever then begins reading the telemetry files and serving the EDUs to the Telemetry process. When the replay is complete, the Request Manager sends the completion status back to the Request Queue Manager. The Request Queue Manager passes the completion status to FUI and FUI then issues a cancel request to the FoDsRequestQueueMgr. FoDsRequestQueueMgr sends the request on to the Request Manager so that the Request Manager can clean up. FoDsReqeustQueueMgr then deletes the request from the request queue.

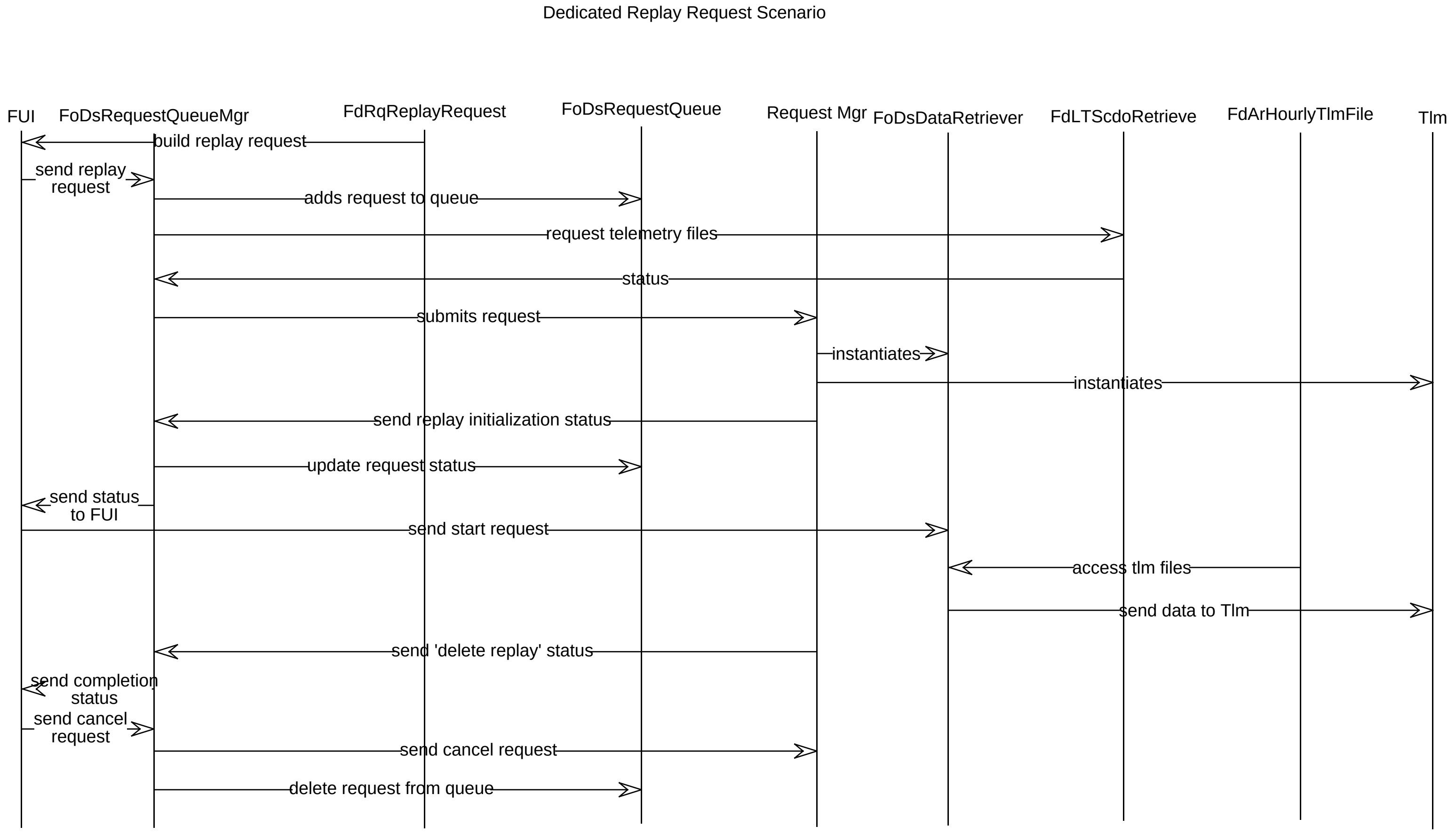


Figure 3.12-4. DMS Telemetry Retrieval Scenario 2 Event Trace

3.12.4.3.1 Telemetry Retrieval/Replay Scenario 3 Abstract

The Telemetry Retrieval/Replay scenario 3 describes the processing of a Shared replay request from FUI.

3.12.4.3.2 Telemetry Retrieval/Replay Scenario 3 Summary Information

Interfaces:

FUI

Analysis

TLM

SCDO

Stimulus:

Receipt of a Shared replay request from FUI.

Desired Response:

Initialize logical string components, enter request onto the queue, & multicast the requested data to whomever is listening.

Upon completion of request, delete entry from the queue.

Pre-Conditions:

Request Queue manager software has been initiated.

Post-conditions:

3.12.4.3.3 Telemetry Retrieval/Replay Scenario 3 Description

The Request Queue Manager receives a shared replay request from FUI. The Request Queue Manager adds the request to the request queue. A quick look is taken to determine the location of the necessary telemetry files. If one or more are located at long-term storage, then FdLTScdoRetrieve is used to request the needed files. A status is returned from SCDO once the transfer is complete. The Request Queue Manager then creates a Data Retriever process on the Data Server which will multicast out the requested telemetry data. The request is then submitted to a Request Manager on an available user station. The Request Manager then initializes the necessary components. The status of this initialization is determined by the Request Manager and sent to the Request Queue Manager. FoDsRequestQueueMgr updates the appropriate entry in the request queue and returns the request status information to FUI.

If the initialization was successful, then FUI sends a start request to the appropriate FoDsDataRetriever process. The Data Retriever then begins reading the telemetry files and multicasting the EDUs to any listening Telemetry process. When the replay is complete, the Request Manager sends the completion status back to the Request Queue

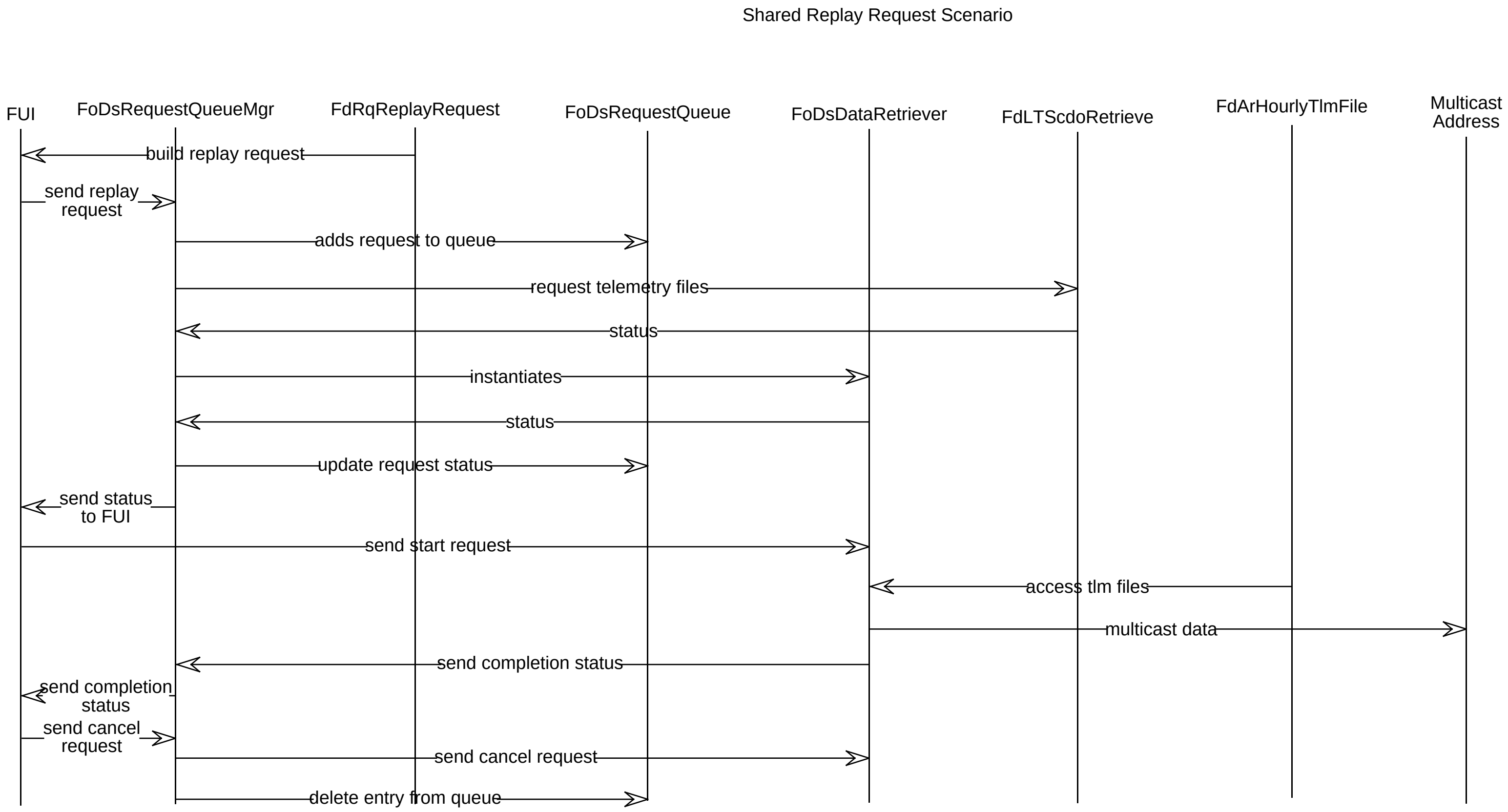


Figure 3.12-5. DMS Telemetry Retrieval Scenario 3 Event Trace

Manager. The Request Queue Manager passes the completion status to FUI and FUI then issues a cancel request to the FoDsRequestQueueMgr. FoDsRequestQueueMgr sends the request on to the Request Manager so that the Request Manager can clean up. FoDsReqeustQueueMgr then deletes the request from the request queue.

3.12.5 DMS Telemetry Retrieval Data Dictionary

FaRpFUIToQueueProxy

class **FaRpFUIToQueueProxy**

This class contains the proxy between FUI and the Request Queue Manager

Public Construction

FaRpFUIToQueueProxy()

This function is the default constructor for the class

~FaRpFUIToQueueProxy()

This function is the default constructor for the class

Public Functions

int **receive**()

This function receives the request

int **send**(FdRqReplayRequest)

This function sends the request

FaRpReqMgrToQueueProxy

class **FaRpReqMgrToQueueProxy**

This class represents the proxy between the queue manager and the request manager

Public Construction

FaRpReqMgrToQueueProxy()

This is the default constructor for the class

~FaRpReqMgrToQueueProxy()

This is the default constructor for the class

Public Functions

int **receive**()

This function receives the messages from the request manager

int **send**(MsgType, ReplayType, FoDsReplayStatus)

This function sends the request status to the request manager. The MsgType can be either: 1) Replay Status, 2) ReqMgr registering, or 3) ReqMgr unregistering with the queue mgr.

The ReplayType parameter is really needed only if the MsgType is 2 or 3.

FaRqDbidLookupProxy

class **FaRqDbidLookupProxy**

This class represents the interface to DMS' DB lookup tool

Public Construction

FaRqDbidLookupProxy()

This is the default constructor for the class

~FaRqDbidLookupProxy()

This is the default destructor for the class

Public Functions

void **receive**()

This function receives the request

int **send**(StartTime, StopTime, Scid)

This function sends the request to DMS

FdArHourlyTlmFile

class **FdArHourlyTlmFile**

This class contains the hourly telemetry file

Public Construction

FdArHourlyTlmFile()

This is the default constructor for the class

~FdArHourlyTlmFile()

This is the default destructor for the class

Public Functions

int **Close**(void)

This function closes the hourly tlm file

int **Open**(void)

This function opens the hourly tlm file

int **Read**(void)

This function reads the hourly tlm file

int **Write**(void)

This function writes to the hourly tlm file

Private Data

char **myFilename**[36]

This member variable contains the tlm file name

FdRqReplayRequest

class **FdRqReplayRequest**

This class represents the information to be passed from FUI to the request queue manager as part of a replay request. This class of data will come over in the FaRpFUIToQueueProxy.

Public Construction

FdRqReplayRequest()

This is the default constructor for the class

~FdRqReplayRequest()

This is the default destructor for the class

Private Functions

enum(Shared, Dedicated, Analysis)

This member variable contains the type of replay being requested (Shared, Dedicated, Analysis)

enum(Submit, Cancel, GetStatus)

This member variable contains the request type (Submit request, Cancel request, Get request status)

Private Data

EcTInt **myDatabaseID**

This member variable contains request database

file* **myParamList**

This member variable contains the request parameter list

EcTInt **myPriority**

This member variable contains the request priority

EcTInt **myReplayRate**

This member variable contains the replay rate

EcTInt **myRequestID**

This member variable contains the request ID

EcTInt **myRequestingProc**

This member variable contains the ID of the requesting FUI

EcTInt **myScid**

This member variable contains the request SCID

EcTTime **myStartTime**

This member variable contains the request start time

EcTTime **myStopTime**

This member variable contains the request stop time

FoDsDataRetriever

class **FoDsDataRetriever**

This class contains the Data Retriever which will retrieve requested telemetry data from the archive.

Public Construction

FoDsDataRetriever()

This function is the default constructor for the class

~FoDsDataRetriever()

This function is the default constructor for the class

Public Functions

int **retrieveTlmSAU**()

This function retrieves the next SAU from the archive

int **sendTlmEDU**()

This function sends the EDU to the Destination address

Private Data

int **myDestinationAddr**

This member variable contains the EDU's destination address

int **myReplayRate**

This member variable contains the replay rate

FoDsRequestQueue

class **FoDsRequestQueue**

This class represents the queue of all requests made to DMS

Public Construction

FoDsRequestQueue()

This is the default constructor for the class

~FoDsRequestQueue()

This is the default destructor for the class

Private Data

queue **entry**

This member variable contains the queue entry of the next portion of the the replay request

queue **entry**

This member variable contains the queue entry of the previous portion of the request (replay requests may be partitioned based on DB crossovers)

IpcAddr* **myDRpid**

This member variable contains the process ID of the Data Retriever which is serving the telemetry

EcTInt **myDatabaseID**

This member variable contains the request DB id

EcTInt **myFileLocation**

This member variable contains the location of the requested data files

file* **myParamList**

This member variable contains the request parameter list

EcTInt **myPriority**

This member variable contains the request priority

EcTInt **myReplayRate**

This member variable contains the request replay rate

EcTInt **myRequestID**

This member variable contains the request ID

EcTInt **myRequestingProc**

This member variable contains the requesting FUI id

EcTInt **myScid**

This member variable contains the request SCID

EcTTime **myStartTime**

This member variable contains request start time

EctInt **myStation**

This member variable contains the station upon which the request is being executed (Analysis requests)

EcTInt **myStatus**

This member variable contains the request status

EcTTime **myStopTime**

This member variable contains the request stop time

EcTInt **myStringID**

This member variable contains the ID of the replay string

FoDsRequestQueueMgr

class **FoDsRequestQueueMgr**

This class represents the request queue manager task

Public Construction

FoDsRequestQueueMgr()

This is the default constructor for the class

~FoDsRequestQueueMgr()

This is the default destructor for the class

Public Functions

FoDsDBTable* **dbLookup**(startTime, stopTime)

This function determines the DB id(s) required to perform the requested replay.

int **deleteFromQueue**()

This function deletes from the request queue

int **findStation**(requestType)

This function finds a user station which is available to perform the requested processing.

int **handleAnaDedReq**()

This function handles analysis & dedicated requests

int **handleSharedReq**()

This function handles shared replay requests

int **init**()

This function initializes the request queue mgr

int **readQueue**()

This function reads the request queue

int **run**()

This executes the request queue manager task

int **updateQueue**()

This function updates the request queue

int **writeQueue**()

This function writes to the request queue

FoRqFUIToDataRetrieverProxy

class **FoRqFUIToDataRetrieverProxy**

This class represents the interface between FUI and the Data Retriever

Public Construction

FoRqFUIToDataRetrieverProxy()

This is the default constructor for the class

~FoRqFUIToDataRetrieverProxy()

This is the default destructor for the class

Public Functions

int **receive**()

This function receives the request

int **send**(RequestType, startTime, stopTime)

This function sends the request to the Data Retriever

Abbreviations and Acronyms

ACL	Access Control List
AD	Acceptance Check/TC Data
AGS	ASTER Ground System
AM	Morning (ante meridian) -- see EOS AM
Ao	Availability
APID	Application Identifier
ARAM	Automated Reliability/Availability/Maintainability
ASTER	Advanced Spaceborne Thermal Emission and Reflection Radiometer (formerly ITIR)
ATC	Absolute Time Command
BAP	Baseline Activity Profile
BC	Bypass check/Control Commands
BD	Bypass check/TC Data (Expedited Service)
BDU	Bus Data Unit
bps	bits per second
CAC	Command Activity Controller
CCB	Change Control Board
CCSDS	Consultative Committee for Space Data Systems
CCTI	Control Center Technology Interchange
CD-ROM	Compact Disk-Read Only Memory
CDR	Critical Design Review
CDRL	Contract Data Requirements List
CERES	Clouds and Earth's Radiant Energy System
CI	Configuration item
CIL	Critical Items List
CLCW	Command Link Control Words
CLTU	Command Link Transmission Unit
CMD	Command subsystem
CMS	Command Management Subsystem
CODA	Customer Operations Data Accounting
COP	Command Operations Procedure
COTS	Commercial Off-The-Shelf
CPU	Central Processing Unit

CRC	Cyclic Redundancy Code
CSCI	Computer software configuration item
CSMS	Communications and Systems Management Segment
CSS	Communications Subsystem (CSMS)
CSTOL	Customer System Test and Operations Language
CTIU	Command and Telemetry Interface Unit (AM-1)
DAAC	Distributed Active Archive Center
DAR	Data Acquisition Request
DAS	Detailed Activity Schedule
DAT	Digital Audio Tape
DB	Data Base
DBA	Database Administrator
DBMS	Database Management System
DCE	Distributed Computing Environment
DCP	Default Configuration Procedure
DEC	Digital Equipment Corporation
DES	Data Encryption Standard
DFCD	Data Format Control Document
DID	Data Item Description
DMS	Data Management Subsystem
DOD	Digital Optical Data
DoD	Department of Defense
DS	Data Server
DSN	Deep Space Network
DSS	Decision Support System
e-mail	electronic mail
Ecom	EOS Communication
ECS	EOSDIS Core System
EDOS	EOS Data and Operations System
EDU	EDOS Data Unit
EGS	EOS Ground System
EOC	Earth Observation Center (Japan); EOS Operations Center (ECS)
EOD	Entering Orbital Day
EON	Entering Orbital Night
EOS	Earth Observing System

EOSDIS	EOS Data and Information System
EPS	Encapsulated Postscript
ESH	EDOS Service Header
ESN	EOSDIS Science Network
ETS	EOS Test System
EU	Engineering Unit
EUVE	Extreme Ultra Violet Explorer
FAS	FOS Analysis Subsystem
FAST	Fast Auroral Snapshot Explorer
FDDI	Fiber Distributed Data Interface
FDF	Flight Dynamics Facility
FDIR	Fault Detection and Isolation Recovery
FDM	FOS Data Management Subsystem
FMEA	Failure Modes and Effects Analyses
FOP	Frame Operations Procedure
FORMATS	FDF Orbital and Mission Aids Transformation System
FOS	Flight Operations Segment
FOT	Flight Operations Team
FOV	Field-Of-View
FPS	Fast Packet Switch
FRM	FOS Resource Management Subsystem
FSE	FOT S/C Evolutions
FTL	FOS Telemetry Subsystem
FUI	FOS User Interface
GB	Gigabytes
GCM	Global Circulation Model
GCMR	Global Circulation Model Request
GIMTACS	GOES I-M Telemetry and Command System
GMT	Greenwich Mean Time
GN	Ground Network
GOES	Geostationary Operational Environmental Satellite
GSFC	Goddard Space Flight Center
GUI	Graphical User Interface
H&S	Health and Safety
H/K	Housekeeking
HST	Hubble Space Telescope

I/F	Interface
I/O	Input/Output
ICC	Instrument Control Center
ICD	Interface Control Document
ID	Identifier
IDB	Instrument Database
IDR	Incremental Design Review
IEEE	Institute of Electrical and Electronics Engineers
IOT	Instrument Operations Team
IP	International Partners
IP-ICC	International Partners-Instrument Control Center
IPs	International Partners
IRD	Interface requirements document
ISDN	Integrated Systems Digital Network
ISOLAN	Isolated Local Area Network
ISR	Input Schedule Request
IST	Instrument Support Terminal
IST	Instrument Support Toolkit
IWG	Investigator Working Group
JPL	Jet Propulsion Laboratory
Kbps	Kilobits per second
LAN	Local Area Network
LaRC	Langley Research Center
LASP	Laboratory for Atmospheric Studies Project
LEO	Low Earth Orbit
LOS	Loss of Signal
LSM	Local System Manager
LTIP	Long-Term Instrument Plan
LTSP	Long-Term Science Plan
MAC	Medium Access Control; Message Authentication Code
MB	Megabytes
MBONE	Multicast Backbone
Mbps	Megabits per second
MDT	Mean Down Time
MIB	Management Information Base

MISR	Multi-angle Imaging Spectro-Radiometer
MMM	Minimum, Maximum, and Mean
MO&DSD	Mission Operations and Data Systems Directorate (GSFC Code 500)
MODIS	Moderate resolution Imaging Spectrometer
MOPITT	Measurements Of Pollution In The Troposphere
MSS	Management Subsystem
MTPE	Mission to Planet Earth
NASA	National Aeronautics and Space Administration
Nascom	NASA Communications Network
NASDA	National Space Development Agency (Japan)
NCAR	National Center for Atmospheric Research
NCC	Network Control Center
NEC	North Equator Crossing
NFS	Network File System
NOAA	National Oceanic and Atmospheric Administration
NSI	NASA Science Internet
NTT	Nippon Telephone and Telegraph
OASIS	Operations and Science Instrument Support
ODB	Operational Database
ODM	Operational Data Message
OMT	Object Model Technique
OO	Object Oriented
OOD	Object Oriented Design
OpLAN	Operational LAN
OSI	Open System Interconnect
PACS	Polar Acquisition and Command System
PAS	Planning and Scheduling
PDB	Project Data Base
PDF	Publisher's Display Format
PDL	Program Design Language
PDR	Preliminary Design Review
PI	Principal Investigator
PI/TL	Principal Investigator/Team Leader
PID	Parameter ID
PIN	Password Identification Number
POLAR	Polar Plasma Laboratory

POP	Polar-Orbiting Platform
POSIX	Portable Operating System for Computing Environments
PSAT	Predicted Site Acquisition Table
PSTOL	PORTS System Test and Operation Language
Q/L	Quick Look
R/T	Real-Time
RAID	Redundant Array of Inexpensive Disks
RCM	Real-Time Contact Management
RDBMS	Relational Database Management System
RMA	Reliability, Maintainability, Availability
RMON	Remote Monitoring
RMS	Resource Management Subsystem
RPC	Remote Processing Computer
RTCS	Relative Time Command Sequence
RTS	Relative Time Sequence; Real-Time Server
S/C	Spacecraft
SAR	Schedule Add Requests
SCC	Spacecraft Controls Computer
SCF	Science Computing Facility
SCL	Spacecraft Command Language
SDF	Software Development Facility
SDPS	Science Data Processing Segment
SDVF	Software Development and Validation Facility
SEAS	Systems, Engineering, and Analysis Support
SEC	South Equator Crossing
SLAN	Support LAN
SMA	S-band Multiple Access
SMC	Service Management Center
SN	Space Network
SNMP	System Network Mgt Protocol
SQL	Structured Query Language
SSA	S-band Single Access
SSIM	Spacecraft Simulator
SSR	Solid State Recorder
STOL	System Test and Operations Language

T&C	Telemetry and Command
TAE	Transportable Applications Environment
TBD	To Be Determined
TBR	To Be Replaced/Resolved/Reviewed
TCP	Transmission Control Protocol
TD	Target Day
TDM	Time Division Multiplex
TDRS	Tracking and Data Relay Satellite
TDRSS	Tracking and Data Relay Satellite System
TIROS	Television Infrared Operational Satellite
TL	Team Leader
TLM	Telemetry subsystem
TMON	Telemetry Monitor
TOO	Target Of Opportunity
TOPEX	Topography Ocean Experiment
TPOCC	Transportable Payload Operations Control Center
TRMM	Tropical Rainfall Measuring Mission
TRUST	TDRSS Resource User Support Terminal
TSS	TDRSS Service Session
TSTOL	TRMM System Test and Operations Language
TW	Target Week
U.S.	United States
UAV	User Antenna View
UI	User Interface
UPS	User Planning System
US	User Station
UTC	Universal Time Code; Universal Time Coordinated
VAX	Virtual Extended Address
VMS	Virtual Memory System
W/S	Workstation
WAN	Wide Area Network
WOTS	Wallops Orbital Tracking Station
XTE	X-Ray Timing Explorer

This page intentionally left blank.

Glossary

GLOSSARY of TERMS for the Flight Operations Segment

activity

A specified amount of scheduled work that has a defined start date, takes a specific amount of time to complete, and comprises definable tasks.

analysis

Technical or mathematical evaluation based on calculation, interpolation, or other analytical methods. Analysis involves the processing of accumulated data obtained from other verification methods.

attitude data

Data that represent spacecraft orientation and onboard pointing information. Attitude data includes:

- Attitude sensor data used to determine the pointing of the spacecraft axes, calibration and alignment data, Euler angles or quaternions, rates and biases, and associated parameters.
- Attitude generated onboard in quaternion or Euler angle form.
- Refined and routine production data related to the accuracy or knowledge of the attitude.

availability

A measure of the degree to which an item is in an operable and committable state at the start of a "mission" (a requirement to perform its function) when the "mission" is called for an unknown (random) time. (Mathematically, operational availability is defined as the mean time between failures divided by the sum of the mean time between failures and the mean down time [before restoration of function].

availability (inherent) (A_i)

The probability that, when under stated conditions in an ideal support environment without consideration for preventive action, a system will operate satisfactorily at any time. The “ideal support environment” referred to, exists when the stipulated tools, parts, skilled work force manuals, support equipment and other support items required are available. Inherent availability excludes whatever ready time, preventive maintenance downtime, supply downtime and administrative downtime may require. A_i can be expressed by the following formula:

$$A_i = \text{MTBF}/\ (\text{MTBF} + \text{MTTR})$$

Where: MTBF = Mean Time Between Failures

MTTR = Mean Time To Repair

availability (operational) (A_o)

The probability that a system or equipment, when used under stated conditions in an actual operational environment, will operate satisfactorily when called upon. A_o can be expressed by the following formula:

$$A_o = \text{MTBM}\ /\ (\ \text{MTBM} + \text{MDT} + \text{ST}\ )$$

Where: MTBM = Mean Time Between Maintenance (either corrective or preventive)

MDT = Mean Maintenance Down Time where corrective, preventive administrative and logistics actions are all considered.

ST = Standby Time (or switch over time)

baseline activity profile

A schedule of activities for a target week corresponding to normal instrument operations constructed by integrating long term plans (i.e., LTSP, LTIP, and long term spacecraft operations plan).

build

An assemblage of threads to produce a gradual buildup of system capabilities.

calibration

The collection of data required to perform calibration of the instrument science data, instrument engineering data, and the spacecraft engineering data. It includes pre-flight calibration measurements, in-flight calibrator measurements, calibration equation coefficients derived from calibration software routines, and ground truth data that are to be used in the data calibration processing routine.

command

Instruction for action to be carried out by a space-based instrument or spacecraft.

command and data handling (C&DH)

The spacecraft command and data handling subsystem which conveys commands to the spacecraft and research instruments, collects and formats spacecraft and instrument data, generates time and frequency references for subsystems and instruments, and collects and distributes ancillary data.

command group

A logical set of one or more commands which are not stored onboard the spacecraft and instruments for delayed execution, but are executed immediately upon reaching their destination on board. For the U.S. spacecraft, from the perspective of the EOS Operations Center (EOC), a preplanned command group is preprocessed by, and stored at, the EOC in preparation for later uplink. A real-time command group is unplanned in the sense that it is not preprocessed and stored by the EOC.

detailed activity schedules

The schedule for a spacecraft and instruments which covers up to a 10-day period and is generated/updated daily based on the instrument activity listing for each of the instruments on the respective spacecraft. For a spacecraft and instrument schedule the spacecraft subsystem activity specifications needed for routine spacecraft maintenance and/or for supporting instruments activities are incorporated in the detailed activity schedule.

direct broadcast

Continuous down-link transmission of selected real-time data over a broad area (non-specific users).

EOS Data and Operations System (EDOS) production data set

Data sets generated by EDOS using raw instrument or spacecraft packets with space-to-ground transmission artifacts removed, in time order, with duplicate data removed, and with quality/ accounting (Q/A) metadata appended. Time span or number of packets encompassed in a single data set are specified by the recipient of the data. These data sets are equivalent to Level 0 data formatted with Q/A metadata.

For EOS, the data sets are composed of: instrument science packets, instrument engineering packets, spacecraft housekeeping packets, or onboard ancillary packets with quality and accounting information from each individual packet and the data set itself and with essential formatting information for unambiguous identification and subsequent processing.

housekeeping data

The subset of engineering data required for mission and science operations. These include health and safety, ephemeris, and other required environmental parameters.

instrument

- A hardware system that collects scientific or operational data.
- Hardware-integrated collection of one or more sensors contributing data of one type to an investigation.
- An integrated collection of hardware containing one or more sensors and associated controls designed to produce data on/in an observational environment.

instrument activity deviation list

An instrument's activity deviations from an existing instrument activity list, used by the EOC for developing the detailed activity schedule.

instrument activity list

An instrument's list of activities that nominally covers seven days, used by the EOC for developing the detailed activity schedule.

instrument engineering data

Subset of telemetered engineering data required for performing instrument operations and science processing.

instrument microprocessor memory loads

Storage of data into the contents of the memory of an instrument’s microprocessor, if applicable. These loads could include microprocessor-stored tables, microprocessor-stored commands, or updates to microprocessor software.

instrument resource deviation list

An instrument's anticipated resource deviations from an existing resource profile, used by the EOC for establishing TDRSS contact times and building the preliminary resource schedule.

instrument resource profile

Anticipated resource needs for an instrument over a target week, used by the EOC for establishing TDRSS contact times and building the preliminary resource schedule.

instrument science data

Data produced by the science sensor(s) of an instrument, usually constituting the mission of that instrument.

long-term instrument plan (LTIP)

The plan generated by the instrument representative to the spacecraft's IWG with instrument-specific information to complement the LTSP. It is generated or updated approximately every six months and covers a period of up to approximately 5 years.

long-term science plan (LTSP)

The plan generated by the spacecraft's IWG containing guidelines, policy, and priorities for its spacecraft and instruments. The LTSP is generated or updated approximately every six months and covers a period of up to approximately five years.

long term spacecraft operations plan

Outlines anticipated spacecraft subsystem operations and maintenance, along with forecasted orbit maneuvers from the Flight Dynamics Facility, spanning a period of several months.

mean time between failure (MTBF)

The reliability result of the reciprocal of a failure rate that predicts the average number of hours that an item, assembly or piece part will operate within specific design parameters. (MTBF=1/(l) failure rate; (l) failure rate = # of failures/operating time.

mean down time (MDT)

Sum of the mean time to repair MTTR plus the average logistic delay times.

mean time between maintenance (MTBM)

The mean time between preventive maintenance (MTBPM) and mean time between corrective maintenance (MTBCM) of the ECS equipment. Each will contribute to the calculation of the MTBM and follow the relationship: 1/MTBM = 1/MTBPM + 1/ MTBCM

mean time to repair (MTTR)

The mean time required to perform corrective maintenance to restore a system/equipment to operate within design parameters.

object

Identifiable encapsulated entities providing one or more services that clients can request. Objects are created and destroyed as a result of object requests. Objects are identified by client via unique reference.

orbit data

Data that represent spacecraft locations. Orbit (or ephemeris) data include: Geodetic latitude, longitude and height above an adopted reference ellipsoid (or distance from the center of mass of the Earth); a corresponding statement about the accuracy of the position and the corresponding time of the position (including the time system); some accuracy requirements may be hundreds of meters while other may be a few centimeters.

playback data

Data that have been stored on-board the spacecraft for delayed transmission to the ground.

preliminary resource schedule

An initial integrated spacecraft schedule, derived from instrument and subsystem resource needs, that includes the network control center TDRSS contact times and nominally spans seven days.

preplanned stored command

A command issued to an instrument or subsystem to be executed at some later time. These commands will be collected and forwarded during an available uplink prior to execution.

principal investigator (PI)

An individual who is contracted to conduct a specific scientific investigation. (An instrument PI is the person designated by the EOS Program as ultimately responsible for the delivery and performance of standard products derived from an EOS instrument investigation.).

prototype

Prototypes are focused developments of some aspect of the system which may advance evolutionary change. Prototypes may be developed without anticipation of the resulting software being directly included in a formal release. Prototypes are developed on a faster time scale than the incremental and formal development track.

raw data

Data in their original packets, as received from the spacecraft and instruments, unprocessed by EDOS.

- Level 0 – Raw instrument data at original resolution, time ordered, with duplicate packets removed.
- Level 1A – Level 0 data, which may have been reformatted or transformed reversibly, located to a coordinate system, and packaged with needed ancillary and engineering data.
- Level 1B – Radiometrically corrected and calibrated data in physical units at full instrument resolution as acquired.
- Level 2 – Retrieved environmental variables (e.g., ocean wave height, soil moisture, ice concentration) at the same location and similar resolution as the Level 1 source data.
- Level 3 – Data or retrieved environmental variables that have been spatially and/or temporally resampled (i.e., derived from

real-time data

Data that are acquired and transmitted immediately to the ground (as opposed to playback data). Delay is limited to the actual time required to transmit the data.

reconfiguration

A change in operational hardware, software, data bases or procedures brought about by a change in a system's objectives.

SCC-stored commands and tables

Commands and tables which are stored in the memory of the central onboard computer on the spacecraft. The execution of these commands or the result of loading these operational tables occurs sometime following their storage. The term “core-stored” applies only to the location where the items are stored on the spacecraft and instruments; core-stored commands or tables could be associated with the spacecraft or any of the instruments.

scenario

A description of the operation of the system in user’s terminology including a description of the output response for a given set of input stimuli. Scenarios are used to define operations concepts.

segment

One of the three functional subdivisions of the ECS:

CSMS -- Communications and Systems Management Segment

FOS -- Flight Operations Segment

SDPS -- Science Data Processing Segment

sensor

A device which transmits an output signal in response to a physical input stimulus (such as radiance, sound, etc.). Science and engineering sensors are distinguished according to the stimuli to which they respond.

- Sensor name: The name of the satellite sensor which was used to obtain that data.

spacecraft engineering data

The subset of engineering data from spacecraft sensor measurements and on-board computations.

spacecraft subsystems activity list

A spacecraft subsystem's list of activities that nominally covers seven days, used by the EOC for developing the detailed activity schedule.

spacecraft subsystems resource profile

Anticipated resource needs for a spacecraft subsystem over a target week, used by the EOC for establishing TDRSS contact times and building the preliminary resource schedule.

target of opportunity (TOO)

A TOO is a science event or phenomenon that cannot be fully predicted in advance, thus requiring timely system response or high-priority processing.

thread

A set of components (software, hardware, and data) and operational procedures that implement a function or set of functions.

thread, *as used in some Systems Engineering documents*

A set of components (software, hardware, and data) and operational procedures that implement a scenario, portion of a scenario, or multiple scenarios.

toolkits

Some user toolkits developed by the ECS contractor will be packaged and delivered on a schedule independent of ECS releases to facilitate science data processing software development and other development activities occurring in parallel with the ECS.